UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE GRADO

Diseño e Implementación de una aplicación para la gestión del Practicum en la facultad de Educación de la UAH

Autor

David Beato Álvaro

Director

Germán Ros Magán

Antonio José de Vicente Rodríguez

Presidente:

Vocal 1º:

Vocal 2º:

CALIFICACIÓN: FECHA:

A mi Familia y al Deporte

INDICE

INDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

1.1. RESUMEN

El presente trabajo fin de grado tiene como objetivo el diseño, creación, desarrollo e implementación de una aplicación que gestione de forma eficiente los Practicum de la Facultad de Educación de la UAH.

Para esa tarea la aplicación recibirá (en formato Excel):

- Datos acerca de los **centros** de la comunidad de Madrid y de Guadalajara, con las plazas que oferta cada uno de los centros escolares para los Practicum y su especialidad, como Audición y lenguaje, Lengua Extranjera, Educación musical, etc.
- Datos acerca de los **alumnos** matriculados ese año en la universidad en los Practicum de los grados de magisterio en educación primaria, educación infantil y en doble grado en infantil y primaria, con información acerca del Practicum matriculado, nota media, etc.
- Datos acerca de las **solicitudes** de los alumnos (X elecciones por orden de preferencia con la oferta en el centro concreto que la que deseen).
- Datos acerca de los **profesores** de la UAH proporcionada por los departamentos, indicando el número de alumnos de Practicums a tutorizar, zonas de preferencia en las que hacerlo, etc).

La aplicación triangulará esta información referente a alumnos, centros y tutores y realizará la asignación optimizando lo máximo posible el resultado, y lo exportará el resultado tanto a algunas de las tablas de la base de datos como a varios Excel en los que presentará la asignación de forma organizada, clara y concisa.

Lenguaje de Programación: Java sobre Netbeans.

Base de datos: phpMyAdmin sobre XAMPP, con MySQL.

1.2. SUMMARY

This final grade work is to design, create, develop and implement an application that efficiently manages the Practicum of the Faculty of Education of the UAH.

For this task the application will receive (in Excel format):

- Information about the **centers** of the community of Madrid and Guadalajara, with the places offered by each school for the Practicum and its specialty, such as Audition and language, Foreign Language, Music Education, etc.
- Information about the **students** enrolled that year in the University in the levels of Teaching in primary, with information about the enrolled Practicum, average grade, etc.
- Information about the student's **request** (X choices in order of preference with the offer in the specific center that they want).
- Information about the UAH **teachers** provided by the departments, indicating the number of students of Practicums to tutor, areas of preference in which to do, etc.).

The application will triangulate this information regarding student's centers and tutors and will perform the assignment optimizing as much as possible the result, and will export the result to some of the tables in the database as well as to various Excel in which it will present the allocation in an organized, clear and concise manner.

Programming language: Java over Netbeans.

Database: phpMyAdmin over XAMPP, with MySQL.

1.3. RESUMEN EXTENDIDO

Todos los años, al comienzo de los periodos de matriculación en la facultad de Educación de la UAH aparece un grave problema de organización, que debe ser solucionado por una o varias personas durante un periodo mínimo de 2 o 3 meses, de forma que se tiene a varias personas haciendo un trabajo cuya gestión se vuelve realmente tediosa puesto que el proceso es a mano y no hay fijado unas normas concretas como se pretende establecer ahora con los algoritmos de asignación que van a implementarse.

Este problema se refiere a la asignación de Practicum de los alumnos de dicha facultad. Es ésta por tanto la razón principal de la realización de este proyecto, de forma que la futura aplicación realizará de forma automática en pocos minutos el trabajo que antes era desarrollado en varios meses por varias personas de forma manual, y garantizando además que el proceso de asignación se realice de forma eficiente, puesto que la asignación manual por parte de profesores siempre puede dar lugar a errores humanos. El presente trabajo fin de grado tiene por tanto como objetivo el diseño, creación, desarrollo e implementación de una aplicación que gestione de forma eficiente la los Practicum de la Facultad de Educación de la UAH. A continuación explicaremos el proceso de forma más detallada, analizando los datos que recibe la aplicación. Se recibirá (en formato Excel):

- **Centros**
  - CAM (Comunidad Autónoma de Madrid):
    Recibirá todos los centros de la comunidad que ofertan plazas para realizar las prácticas. En el formato Excel empleado, cada centro registrará de 1 a N filas en el documento, representando cada una de las filas un tipo de oferta (indicando la especialidad, el tutor asignado, si es bilingüe, la zona utilizada, etc.).
  - Guadalajara:
    Al igual que la comunidad de Madrid, recibirá el mismo tipo de información, la diferencia radica en que el formato aquí será diferente, puesto que cada fila del documento representa el centro, y en esa misma fila ya se indican la

totalidad de las plazas que ofertan (por ejemplo 3 de Educación Primaria, 5 de Educación Musical, etc.).

- **Alumnos**

  Se recibirá una larga lista de documentos, que tendrán información acerca de cada alumno como el grado, el Practicum matriculado, el nombre del alumno, su DNI, y su nota media (que será importante ya que los que antes eligen son los que mejor nota media tienen). Así pues el número de documentos que se recibirán de los alumnos será uno por cada Grado:

  - Grado de Magisterio en Educación Infantil.
  - Grado de Magisterio en Educación Primaria
  - Doble Grado de magisterio en Educación Infantil–Primaria.

  Y para cada Grado, uno por cada Practicum:

  - Practicum I
  - Practicum II
  - Practicum III
  - Practicum III - Conocimiento del Entorno
  - Practicum III - Lengua Extranjera
  - Practicum III - Lengua y Literatura Española
  - Practicum III – Necesidades educativas especiales
  - Practicum III – Expresión Artística Integral

  Por tanto si se recibiesen la totalidad de los archivos disponibles se recibirán un total de 3 * 8 = 24 archivos con esta información.

- **Solicitudes**

  Al empezar el curso, los alumnos, en un sistema ajeno, se les da la posibilidad de marcar diversas elecciones por orden de preferencia (por ejemplo 10 elecciones), indicando en dicha elección la plaza que quieren y en qué centro la quieren. Cabe mencionar que a la hora de asignar esos alumnos a las plazas solicitada, como se comentará más adelante, se adjudicará primero a los alumnos de P3 (de más a menos calificación), luego a los de P1 (de más a menos calificación), y por último los de P2 (de más a menos calificación).
  Los alumnos que al finalizar el proceso no hayan sido asignados porque ninguna de las X elecciones que hicieron se encuentra disponible, serán sacados en un Archivo Excel para proceder a su

asignación manual por parte de alguien de la facultad.

- **Profesores**

  De los diferentes departamentos de profesores se recibirán archivos con el número de Practicums a tutorizar por cada uno de los profesores, que tipo de Practicum van a tutorizar, las zonas[1] de preferencia, etc.

La aplicación primero realizará la asignación de los alumnos y en los centros. Para ello cargará toda la información de los alumnos y sus notas medias procedentes de los 18 archivos anteriormente comentados, y los cargará en la base de datos. Acto seguido cargará las ofertas ofertadas por cada uno de los centros de Madrid y de Castilla la Mancha, y por último cargará la información de las elecciones de los alumnos por orden de preferencia. Con toda esa información, la aplicación cogerá primero los alumnos de PIII, y consultará uno por uno (de más a menos calificación) las elecciones que ha realizado, asignando la primera disponible, y pasando al siguiente alumno. El mismo procedimiento se seguirá para PI y para PII (en este orden).

Cabe indicar que un alumno con un 8,4 matriculado por ejemplo en PIII, va a tener preferencia a la hora de ser asignado frente a uno que tenga un 9,7 pero esté matriculado por ejemplo en PI o en PII (esto se ha diseñado así porque los alumnos de PIII se refieren a especialidades y en teoría son más difíciles de encuadrar en la asignación).

Una vez asignados todos los alumnos (o la mayoría, sacando los no asignados en un Excel), se procederá a la asignación con los tutores, mostrando especial atención a la lista de zonas en las que el profesor puede impartir el Practicum y contrastándolas con la zona a la que pertenece cada centro (indicadas en los archivos de Guadalajara y de la comunidad de Madrid).

Con la asignación Alumno-Oferta ya realizada, ahora se asignará a cada tutor el número de ofertas que ellos hayan deseado tutorizar en su archivo del

[1] Zona: Para el presente trabajo, la comunidad de Madrid y Guadalajara ha sido dividida en diversas zonas para realizar la asignación de tutores y alumnos de forma eficiente evitando lo máximo posible grandes desplazamientos a la hora de realizar los Practicums.

departamento de profesores. Esta nueva asignación de profesores puede realizarse con 2 algoritmos, cuyo procedimiento a seguir es idéntico, lo único que cambia es el orden en el que son asignados los profesores:

- Algoritmo **suma**: Los primeros profesores en elegir serán aquellos que tengan un mayor número de Practicum en total a impartir, y por último los que menos. En caso de empate entre dos profesores que impartan mismo número de practicums, se tendrá en cuenta el orden alfabético de la primera letra del apellido (marcando como letra preferente una escogida por el usuario de la A la Z).
- Algoritmo **prioridad**: En el Excel a introducir habrá una columna llamada prioridad que indique el orden en que serán asignados los profesores de 1 a N, siendo 1 el primero en elegir y N el último.

Una vez establecido el orden en que serán asignados los profesores, se prestará atención a la elección de éstos, en la que pueden elegir el número de PI, PII y PIII que quieren tutorizar, indicando en el caso de PIII los dos códigos de practicums de las especialidades que pueden ser asignados.

Así pues, la aplicación recorrerá todas y cada una de las zonas que el profesor indicó en las zonas de preferencia, y buscará si en ellas hay algún centro que tenga un número mayor o igual de ofertas de PI, de PII, y de PIII que las que el profesor desea tutorizar, de forma que ese profesor solo tenga que desplazarse durante el curso a un solo centro. Si no hay ningún centro en el que se pueda asignar íntegramente a un profesor, entonces el algoritmo buscara en las zonas aquellos centros que tienen mayor número de ofertas por tutorizar, de manera que el profesor, ya que no ha podido ser asignado en un centro únicamente, sea asignado en el menor número de colegios posibles. En caso de que el profesor no haya podido ser asignado ni repartido en diferentes centros, se exportará un Excel con los profesores que han sido asignados hasta ese momento, y los que no han sido asignados, de forma que el usuario proceda a su asignación manual.

Después de esta pausa temporal de la ejecución, el usuario procederá a introducir de forma manual el profesor que la aplicación no ha sido capaz de asignar (observando para ello los ficheros exportados). Una vez introducido ese profesor "problemático" el usuario podrá volver a ejecutar el algoritmo para que prosiga la asignación de forma automática.

Cabe prestar especial atención al PIII generalista. Todos los profesores pueden impartir tanto el PIII específico que han indicado, como el generalista, de forma que a la hora de buscar ofertas, se seleccionar aquellas que tengan o bien uno de los dos códigos específicos que indicó el profesor, o bien el código del PIII generalista. Lo importante realmente a destacar es que una oferta con un alumno matriculado en código generalista, puede ser asignado un tutor de una especialidad sin ningún problema, pero lo que no puede ocurrir es que un alumno haya sido matriculado en un PIII concreto, como por ejemplo Educación Física, y le sea asignado un profesor que solo imparte PIII generalistas, o que imparte PIII concretos, pero son de otra especialidad, como Pedagogía terapéutica. (el caso de los generalistas se explica más adelante).

Para el desarrollo de la aplicación se ha programado en lenguaje Java utilizando la plataforma Netbeans debido a su gran facilidad para crear la interfaz de cara al usuario. La base de datos en la que se irán almacenando los resultados será phpMyAdmin, sobre XAMPP, y en lenguaje SQL. Se escogió esta base de datos debido a la gran facilidad de interacción con Java y con Netbeans, y porque es un lenguaje de base de datos estandarizado para que en un futuro exista la posibilidad de realizar modificaciones en la aplicación de forma sencilla.

1.4. EXTENDED SUMMARY

Every year, the beginning of the enrollment periods in the UAH faculty of Education appears a serious problem of organization, that must be solved by several people during a minimum period of 2 or 3 months, so has several people doing a job with management makes it really tedious since the process is a hand and there are no specific rules as set out now with the allocation algorithms that are going to be implemented.

This problem refers to the assignment of Practice of students of the same faculty. It is therefore the main reason for the realization of this project, in the way that the future application made automatically in a few minutes the work that was previously developed in various months by several people manually, and also ensuring that the process of Assignment is done efficiently, since manual assignment by teachers can always lead to human error. The aim of this final grade work is therefore the design, creation, development and implementation of an application that is managed efficiently in the Practice of the Faculty of Education of the UAH. The process is shown in more detail, analyzing the data received by the application. You will receive (in Excel format):

- **Centers**
  - CAM (Autonomous Community of Madrid):

    You will receive all the centers in the community that offer places to practice. In the Excel format used, each center registers 1 to N rows in the document, each row representing a type of offer (indicating the specialty, the assigned tutor, if bilingual, the area used, etc.).
  - Guadalajara:

    Like the community of Madrid, received the same type of information, the difference is that the format here is different, since each row of the document represents the center, and in that same row and indicates all the places that They offer (for example 3 of Primary Education, 5 of Musical Education, etc.).

- **Students**

  You will receive a long list of documents, which will have information about each student such as the degree, the registered Practicum, the name of the student, his ID, and his average grade (which will be important and which the Media have). Thus the number of documents received from children is one for each Degree:

  - Grado en Magisterio en Educación Infantil.
  - Grado en Magisterio en Educación Primaria
  - Doble Grado en magisterio de Educación Infantil–Primaria.

  And for each Degree, one for each Practicum:

  - Practicum I
  - Practicum II
  - Practicum III
  - Practicum III - Conocimiento del Entorno
  - Practicum III - Lengua Extranjera
  - Practicum III - Lengua y Literatura Española
  - Practicum III – Necesidades educativas especiales
  - Practicum III – Expresión Artística Integral

  Therefore, if they receive all the available files, they receive a total of 3 * 8 = 24 files with this information.

- **Requests**

  At the beginning of the course, the students, in a foreign system, the options to mark several elections in order of preference (for example 10 elections), indicating in the choice of the place they want and in what city center. It should be mentioned that when assigning these students to the requested places, as discussed below, students will first be awarded P3 (after a lower grade), then P1, and finally P2 (From more to less qualification).

  Students who complete the process without the system were assigned because none of the X choices they made are available; they are removed in an Excel File for the procedure of their manual assignment by someone from the faculty.

- **Teachers**

  The different departments of teachers will receive the files with the number of Practicums a tutorial for each of the teachers, which type of Practicum a tutoring, preference areas[2], etc.

The application made in the assignment of the students and in the centers. For the loader in the database all the information of the students and the average notes coming from the 18 files previously commented, and the loaders in the database. He then uploaded the data offers for each of the Madrid center and the apple to the database, and finally uploaded the information of the students' election in order of preference to the database. With all this information, the application will first take the P3 students, and consult one by one (over grade less) the choices that have made, assigning the first available, and passing to the next student. The same procedure is followed for PI and for PII (in this order).

It should be noted that the student with 8.4 enrolled for example in PIII, will have preference when it is assigned to one that has a 9.7 but is enrolled for example in PI or PII As the students of PIII Refer to the specialties and in the theory of the most difficult to fit in the assignment).

Once all the students have been assigned (most of them, taking out the none assigned in an Excel), the assignment with the drivers is processed, showing special attention to the list of areas in which the teacher can impart the Practicum and contrast with the Area to which each center belongs (indicated in the archives of Guadalajara and the community of Madrid).

With the Student-Offer assignment already made, each tutor will now be assigned the number of offers that they have wanted to tutor in their teacher department file. This new assignment of teachers can be done with 2 algorithms, whose procedure to follow is identical; the only thing that changes is the order in which the teachers are assigned:

[2] Areas: For the present work, the community of Madrid and Guadalajara has been divided in several zones to realize the assignment of tutors and students of efficient form avoiding to the maximum possible great displacements when doing the Practicums.

- Algorithm **sum**: The first teachers to choose will be those who have a greater number of Practicum in total to impart, and lastly the less. In the case of a tie between two teachers who tutor the same number of practicums, the alphabetical order of the first letter of the surname (marking as preferred letter a chosen by the user of the A to Z) will be taken into account.
- Algorithm **priority**: In the Excel to enter there will be a column called priority that indicates the order in which the teachers will be assigned from 1 to N, with 1 being the first to choose and N being the last.

Once the order in which the teachers are assigned, attention will be given to the choice of teachers, in which they can choose the number of PI, PII and PIII they want to tutor, indicating in the case of PIII the two codes of practicums The specialties that can be assigned.

Thus, the application will go through each and every one of the areas that the teacher indicated in the preference zones, and will look for if there is a center that has a greater or equal number of IP, PII, and PIII offers than Which the teacher wishes to tutor, so that this teacher only has to move during the course to a single center. If there is no center in which a teacher can be assigned in its entirety, then the algorithm will search in the zones those centers that have the greatest number of tutorial offers, so that the teacher, since he could not be assigned in a center Only, be assigned in the fewest possible schools. In case the teacher cannot be assigned or distributed in different centers, an Excel will be exported with the teachers that have been assigned until that moment, and those that have not been assigned, so that the user proceeds to its manual assignment.

After this temporary pause of execution, the user will proceed to manually enter the teacher that the application has not been able to allocate (observing for the exported files). Once this "problematic" teacher has been introduced, the user can re-execute the algorithm to continue the assignment automatically.

Particular attention should be paid to the general PIII. All teachers can teach both the specific PIII they have indicated and the generalist, so that when searching for offers, select those that have either one of the two specific codes indicated by the teacher, or the code PIII generalist. The important thing to note is that an offer with a student enrolled in generalist code can be assigned a tutor of a

specialty without any problem, but what can not happen is that a student has been enrolled in a particular PIII, such as Physical Education, and be assigned a teacher who only imparts PIII generalists, or who imparts concrete PIII, but are of another specialty, such as Therapeutic Pedagogy. (The case of the generalists is explained later).

For the development of the application has been programmed in Java language using the Netbeans platform due to its great facility to create the interface facing the user. The database in which the results will be stored will be phpMyAdmin, about XAMPP, and in SQL language. This database was chosen because of the great ease of interaction with Java and Netbeans, and because it is a standardized database language so that in the future there is the possibility of making modifications to the application in a simple way.

1.5. PALABRAS CLAVE

Java, MySQL, alumnos, centros, profesores.

1.6. KEYWORDS

Java, MySQL, students, centers, teachers.

1.7. OBJETIVOS DEL PROYECTO

Al inicio del curso académico aparece la ardua tarea de asignar manualmente alumnos, ofertas y profesores, para la que es necesario que varias personas empleen gran cantidad de tiempo.

Con el presente proyecto se pretende crear una aplicación sencilla, intuitiva, y que realice la triangulación de toda la cantidad de información necesaria, para realizar la asignación de Practicum de forma rápida y eficiente.

Como dato mencionar que actualmente la cantidad aproximada de datos a trabajar es la siguiente:

- 200 centros
- 1458 ofertas
- 748 alumnos
- 60 profesores

1.8. PROJECT OBJECTIVES

At the beginning of the academic year the arduous task of manually assigning students, offers and teachers, for which it is necessary that several people use a great amount of time.

This project aims to create a simple, intuitive application, and to triangulate the amount of information needed to perform the assignment of Practicum quickly and efficiently. As can be mentioned, the approximate amount of data to work is currently as follows:

- 200 centers
- 1458 offers
- 748 students
- 60 teachers

2. HERRAMIENTAS UTILIZADAS

2.1. TOAD DATA MODELER

2.1.1. INTRODUCCION

Toad Data Modeler se trata de una herramienta fácil de usar que ayuda a las organizaciones a crear, mantener, y documentar sus sistemas de base de datos con una interfaz gráfica sencilla de utilizar.

Anteriormente a este programa se le denominaba “CASE Studio 2”, antes de ser adquirida desde Charonware por Quest Software en 2006 y Quest Software fue adquirida por Dell a partir del año 2012.

Con esta herramienta, se puede por tanto crear estructuras de bases de datos (lógico y físico, Diagramas entidad relación, etc.). En este proyecto se ha utilizado para generar el diagrama Entidad-Relación de forma gráfica, para luego utilizar otra de las aplicaciones de este programa, que es la generación de forma automática del código SQL para la base de datos, que será el Script que luego importaremos en phpMyAdmin como se explicará más adelante.

Así pues, es una aplicación que no sólo permite diseñar esquemas de base de datos, sino también generar el código SQL necesario para producirlas. Con él puedes desarrollar diagramas para la mayor parte de sistemas gestores de bases de datos existentes, como son MySQL, Access, etc. Además, la aplicación resulta muy útil también a la hora de generar detallados informes en HTML y RTF.

2.1.2. CARACTERÍSTICAS

Estas son algunas de las principales características que presenta:

- **Diseño**: Permite crear estructuras de alta calidad, generarlas

automáticamente, siguiendo los estándares de buenas prácticas y metodologías de diseño más avanzadas.

- **Documentar:** Permite generar reportes detallados para documentar fácilmente las bases de datos existentes.
- **Rediseñar**: Puede tomar una base de datos, rediseñar el modelo y generar el SQL del nuevo diseño.
- **Migrar**: Permite trasladar en forma simple las estructuras de una base de datos a otra versión o plataforma a la hora de realizar migraciones.
- **Sincronizar**: Permite comparar un modelo con una base de datos existente.

2.1.3. VENTAJAS E INCONVENIENTES

Sin embargo, aunque las ventajas que presenta esta aplicación son las que se han mencionado en el anterior apartado, también presenta algunos inconvenientes, como que no hay versiones de Toad para sistemas operativos UNIX/Linux ni para Mac OS, puesto que únicamente funciona bajo Windows.

Mencionar también como ventaja que tiene compatibilidad con cuatro de los más populares gestores de bases de datos que existen en la actualidad, que son Oracle Database, Microsoft SQL Server, IBM DB2 y MySQL. Presenta también una gran comunidad de usuarios que ofrecen mucha ayuda acerca de cómo aprender a usar este programa y encontrar soluciones para problemas particulares.

2.2. MYSQL

2.2.1. INTRODUCCIÓN

SQL son las siglas de Structured Query lenguaje, y es el lenguaje estandarizado más común para acceder a bases de datos. MySQL se trata de uno de los sistemas de gestión de bases de datos SQL de código abierto más populares. Cuando hablamos de código abierto, nos referimos que cualquier puede utilizar y modificar el software, es decir, descargárselo de internet de forma gratuita y estudiar y cambiar el código de acuerdo con sus necesidad. El hecho de que sea de código abierto no es lo mismo que Software libre. Aunque nació inicialmente como una iniciativa de Software libre y aún sigue ofreciéndose como tal, para usuarios particulares, si se desea emplearlo para gestionar datos de una empresa, se puede comprar una licencia, como un software propietario, que es la autoría de la empresa que lo patrocina.

La licencia que utiliza se trata una licencia GPL (GNU General Public License), que define lo que se puede y no se puede realizar con el software en diferentes contextos. Aún así, como decíamos en el párrafo anterior, si queremos utilizar el software con fines comerciales y la primera licencia no es suficiente, se puede adquirir una licencia comercial con la cual la compañía ofrece soporte y servicios añadidos.

La mayor parte del código se encuentra escrito en lenguaje C/C++ y la sintaxis de su uso es bastante sencilla, y ello permite crear bases de datos simples o complejas con mucha facilidad, teniendo además la ventaja de su compatibilidad con diferentes plataformas informáticas, y ofreciendo una infinidad de aplicaciones que permiten acceder rápidamente a las sentencias del gestor de base de datos.

A día de hoy se puede transformar cualquier proyecto web en la necesidad de utilizar grandes cantidades de información de forma estructurada. Su importancia radica en que para procesar o acceder información almacenado en una base de datos, se necesita un gestor de base de datos como es el que estamos analizando.

Se trata también de un gestor de base de datos en multihilo y multiusuario, esto es, puede ser utilizado por varias personas al mismo tiempo, e incluso, realizar varias consultas en el mismo tiempo, de ahí que sea un sistema tan versátil y que se haya hecho tan popular en los últimos tiempos. Esta característica es la que provoca que pueda trabajar en sistemas cliente/servidor o en sistemas empotrados.

Se trata de bases de datos relacionales, esto es las tablas utilizadas en la base de datos no van a almacenar una gran cantidad de información de diferente naturaleza, sino que se organizan conteniendo información que describe un objeto concreto dentro del sistema, y después se pueden establecer relaciones entre diferentes tablas. Estas relaciones, como explicaremos más adelante (puesto que son las que nosotros hemos empleado) serán 1:1, 1:N, N:M, identificativas, no identificativas, 0:N, 0;1, etc.

Mencionar también que el servidor de base de datos es muy rápido, fiable, escalable y muy sencillo de utilizar. Puede también ser ejecutado en ordenadores y portátiles junto a otras aplicaciones, servidores, etc. puesto que no requiere un gran consumo de recursos para su funcionamiento. Estas características hacen que sea un sistema gestor de base de datos muy utilizado en desarrollo web, ya que permite a los desarrolladores y diseñadores, realizar cambios en sus sitios de manera simple, con tan sólo cambiar un único archivo, evitando el tener que modificar todo el código web (de debe a que MySQL trabaja con un sistema centralizado que permite realizar cabios en un solo archivo y que se haga un cambio en toda la estructura de la base de datos).

En cuanto al contexto histórico, su origen se remonta a la década de los 80 en la que no se encontraba un sistema de almacenamiento para ficheros que resultase eficiente y “Michael Widenius” decidió diseñar su propio sistema. Así pues, inicialmente fue desarrollado por MySQL AB para poder manejar grandes bases de daros mucho más rápido de lo que lo hacían las soluciones existentes y se ha utilizado con éxito a lo largo de muchos años en entornos exigentes. Dicha empresa fue comprada más tarde por Sun Microsystems, siendo esta a su vez

comprada por Oracle en 2010.

Decir que aunque en su creación inicialmente carecía de elementos esenciales hoy en día como la integridad referencial o las transacciones, esas características fueron añadidas más tarde por los programadores debido a las necesidades de la sociedad, así como muchos otros tipos de carencias.

Como conclusión, cabe mencionar que se encuentra actualmente en constante desarrollo, que posee una gran cantidad de funciones, y que su conectividad, velocidad y seguridad hacen que se trate por tanto de una de los más populares sistemas de gestión de base de datos más apropiados para acceder a bases de datos en Internet.

2.2.2. CARACTERÍSTICAS

MySQL posee una gran cantidad de características que han hecho que sea el más importante gestor de base de datos de todos los tiempos, algunas son:

- Multihilo: Gracias a ello aprovecha la potencia de sistemas multiprocesadores.
- Gran cantidad de tipos de datos para las columnas. (float, double, char, text, varchar, etc.).
- Soporta hasta 32 índices por tabla.
- Gran portabilidad entre sistemas.
- Se puede obtener su código fuente vía Internet. Gran parte de su desarrollo se debe a esta característica, puesto que programadores de todo el mundo han trabajado durante los últimos años para añadirle características que multipliquen su potencial.
- MySQL es el más rápido gestor de base de datos de Internet.
- Es de fácil instalación y configuración.
- Gran facilidad de uso.
- Su utilización es gratuita.
- APIs disponibles en C, C++, Eiffel, Java, Perl, PHP, Phyton, Ruby y Tcl.

- Proporciona sistemas de almacenamiento transaccional y no transaccional.
- Tiene infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación.
- Tiene un buen nivel de seguridad en los datos mediante la correcta gestión de usuarios y passwords.
- Está escrito en C y C++.
- Tablas hash que son utilizadas como tablas temporales.

2.2.3. VENTAJAS E INCONVENIENTES

En vista de los anteriores apartados ha quedado visto que MySQL es el más importante gestor de base de datos, y esto es debido a la gran cantidad de ventajas que presenta y que ya hemos mencionado, sin embargo, tiene algún pequeño inconveniente. A continuación se presentan las ventajas y los inconvenientes más relevantes.

VENTAJAS

- MySQL software es OPEN Source.
- Gran velocidad al realizar las operaciones y gran rendimiento.
- Fácil configuración e instalación.
- Conectividad y seguridad muy elevada.
- Soporta gran cantidad de sistemas operativos.
- No necesita grandes recursos para ser ejecutado.
- Baja probabilidad de corromper datos, aún con errores propios.

INCONVENIENTES

- Muchas utilidades de MySQL no están documentadas.
- No es intuitivo como otros programas, como por ejemplo ACCESS.
- La función de conversión CAST() no soporta la conversión a REAL o BIGINT.
- Los privilegios de una tabla no se eliminan automáticamente cuando se borra una tabla.

2.2.4. SENTENCIAS Y FUNCIONES

Algunas de las sentencias utilizadas en MySQL son SELECT, INSERT, UPDATE, DELETE, etc. Además también permite hacer una organización de las tuplas obtenidas mediante SQL GROUP BY y ORDER BY, o realizar intersecciones de conjuntos de datos mediante NATURAL INNER JOIN, LEFT INNER JOIN, RIGHT OUTER JOIN, etc.

2.3. PHPMYADMIN SOBRE XAMPP

2.3.1. INTRODUCCIÓN

PhpMyAdmin se trata de una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web utilizando Internet. Actualmente puede crear y eliminar bases de datos con gran facilidad, así como crear, eliminar y alterar tablas, borrar, editar campos y ejecutar cualquier sentencias SQL o administrar privilegios o exportar datos en varios formatos.

Por otro lado, XAMPP es un paquete de instalación independiente de plataforma, de software libre, y que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor apache, y los intérpretes para lenguajes de script: PHP y Perl, distribuyéndose bajo la licencia GPL.

Básicamente XAMPP sirve como una herramienta de desarrollo que te permite probar tu trabajo (páginas web o programación por ejemplo), en tu propio ordenador sin necesidad de tener que acceder a internet.

Así pues, en el presente proyecto, XAMPP lo utilizaremos para acceder a phpMyAdmin en modo localhost, y poder manejar la información en phpMyAdmin de todas las tablas que modificamos mediante Java a medida que va ejecutándose la aplicación y que comentaremos más adelante.

2.3.2. CARACTERÍSTICAS

Algunas de las características más importantes de phpMyAdmin es que es un manejador de base de datos MySQL, MariaDB y Drizzle, que permite la importación de datos desde CSV y SQL (en este proyecto se hará uso de la

importación vía SQL), permite la administración de múltiples servidores, crear gráficos PDF del diseño de la base de datos, crear consultas complejas usando Query-by-exmple (QBE) y permite también la exportación de datos a varios formatos como CSV, SQL, XML, PDF, Word, Excel, y otros.

Por otro lado, las características más importantes de XAMPP es que este sistema es que es multiplataforma, es decir, existen versiones para diferentes sistemas operativos, como Microsoft Windows, GNU, etc. Además es de fácil instalación puesto que solo requiere descargar y ejecutar un archivo, lo que permite ahorrar bastante tiempo.

2.3.3. VENTAJAS E INCONVENIENTES

Se ha utilizado esta herramienta debido a su gran simplicidad de cara al usuario al ser instalada. Además es el programa más utilizado en interacciones Java-MySQL, y en nuestro caso, nos da la posibilidad de que la aplicación pueda interaccionar con PHP en un futuro, de forma que podría hacerse una versión web sin cambiar de herramienta.

2.4. JAVA

2.4.1. INTRODUCCIÓN

Java se trata de un lenguaje de programación y de una plataforma informática comercializada por primera vez en 1995 por Sun MicroSystemes. A día de hoy ha adquirido especial importancia puesto que muchas aplicaciones y sitios web no funcionarán a menos que tenga Java instalado. Es rápido, seguro, fiable, y se utiliza desde centros de datos o consolas hasta súper computadoras, pasando por teléfonos móviles y tabletas.

Se trata de un lenguaje del tipo ORIENTADO A OBJETOS, y que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuese posible. Al ser del tipo orientado a objetos, quiere decir que los datos y el código responden a combinaciones de entidades llamada “objetos”. La principal características de estos lenguajes es que cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa, y es esta separación en objetos coherentes e independientes la que ofrece una mayor estabilidad y eficiencia a la hora de diseñar un sistema software.

En este proyecto se ha elegido Java como lenguaje de programación para desarrollar la aplicación que se va a encargar de interactuar con phpMyAdmin para gestionar la información de alumnos, profesores, etc. a través de una base de datos sobre MySQL. Y el lenguaje Java lo utilizaremos sobre la aplicación Netbeans que comentaremos en el siguiente apartado.

2.4.2. CARACTERÍSTICAS

Entre las características más importantes de este lenguaje de programación no es solo que sea un lenguaje orientado a objetos. Estas son algunas:

- Simple
- Tipado estáticamente
- Distribuido
- Interpretado
- Seguro
- Arquitectura neutral.
- Robusto
- Multihilo
- Recolector de basura (Esta características es muy importante).
- De alto rendimiento (con hardware especializado sobre todo).

Así pues, se trata del lenguaje más utilizado en los últimos tiempos debido a su simplicidad y eficiencia a la hora de ser programado. Está inspirado en muchos anteriores como C o C++ (que también es orientado a objetos).

Así mismo, permite la interacción con la base de datos mediante instrucciones sencillas, que son las que se han utilizado en la aplicación para realizar la gestión, como veremos a continuación.

2.4.3. VENTAJAS E INCONVENIENTES

Este lenguaje de programación tiene la gran ventaja de que es mundialmente conocido y muy sencillo de programar. Así pues, la razón por la que se eligió este lenguaje fue pensando en simplificar una posible futura modificación del código.

2.5. NETBEANS

2.5.1. INTRODUCCIÓN

La aplicación que se desarrollará para el presente trabajo se realizará en Java y dicho lenguaje de programación se ejecutará con la plataforma Netbeans, de ahí que procedamos a comentar las principales características de este compilador.

Netbeans, que fue desarrollado por Sun Microsystems en 2000, se trata de un entorno de desarrollo integrado libre, hecho sobre todo para el lenguaje Java (a diferencia de Eclipse, que acepta mas entornos sin problema). Se trata de un producto libre y gratuito sin restricciones de uso.

Esta plataforma permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes software llamados módulos siendo cada uno de estos módulos un archivo Java que contiene clases de java escritas para interactuar con las API's de Netbeans. Esto presenta la ventaja que las aplicaciones construidas a partir de módulos pueden ser extendidas agregándoles nuevos módulos, es decir, que las aplicaciones basadas en la plataforma Netbeans pueden ser extendidas fácilmente por otros desarrolladores de software.

Así pues, como conclusión, diremos que Netbeans IDE es un entorno de desarrollo integrado (IDE), modular y de base estándar (normalizado), y escrito en el lenguaje de programación orientado a objetos Java. Se trata de un IDE de código abierto y de una plataforma de aplicación que puede ser usado como una estructura de soporte general (framework) para compilar cualquier otro tipo de aplicación.

2.5.2. CARACTERÍSTICAS

Entre las características más importantes de este soporte de programación se encuentran algunas de las siguientes:

- Da soporte a todas las novedades que ocurran en el lenguaje Java.

- Buen asistente para la creación y configuración de distintos proyectos, incluida la elección de algunos frameworks.
- Buen editor de código, multilenguaje, con coloreado y sugerencias de código constantes que hacen la programación más amigable de cara al usuario, así como localización de la ubicación de la clase actual, comprobaciones sintácticas, etc.
- Simplifica la gestión de grandes proyectos con el uso de diferentes vistas, asistentes de ayuda, etc.
- Depurado de errores: Es bastante útil puesto que te señala donde está el error y con solo un click él mismo puede encargarse de solucionarlo automáticamente cambiando el código.
- Permite optimizar el código con su "Profiler", ayudando a que éstas se ejecuten más rápido y con el mínimo uso de memoria.
- Es fácilmente extensible a través de plugins
- Se integra con diversos servidores de aplicaciones de forma que podemos gestionarlos desde el propio IDE, como Apache, Glassfish, etc.

Sin embargo, la característica más importante y que no hemos mencionado, es que desde Netbeans podemos conectarnos a distintos sistemas gestores de bases de datos como Oracle, o en nuestro caso, MySQL, ver las tablas desde el propio IDE, realizar consultas, modificaciones, etc. , siendo esta la principal razón por la que se escogió este entorno de desarrollo, debido a la gran cantidad de interacciones que tiene nuestra aplicación con la base de datos en phpMyAdmin a la hora de gestionar los practicums.

2.5.3. INTERFAZ GRÁFICA

Este entorno de desarrollo se escogió para la realización de este proyecto por dos razones. La primera es la comentada en el párrafo anterior, que es el fácil acceso a la base de datos, permitiendo la visualización, creación y modificación o eliminación de tablas desde el propio entorno. La segunda las razones es debido a la

parte de la interfaz gráfica de la aplicación.

La presente aplicación en un futuro no va a ser ejecutada por programadores, sino por usuarios medios que necesitan un entorno amigable a la hora de insertar los archivos que contengan la información de los alumnos, centros, etc. Así pues, se hace imprescindible la creación de una interfaz gráfica, y Netbeans posee una gran variedad de ventanas, campos de texto, colores, etc. a la hora de crear la interfaz. Así pues, Netbeans es la mejor plataforma en la creación de interfaces gráficas por medio de una interfaz gráfica (valga la redundancia).

Permite la creación de paneles, cambio de imágenes, jugar con las ventanas, etc. Esta parte a desarrollar se verá en al manual de usuario.

2.5.4. VENTAJAS E INCONVENIENTES

El motivo de Netbeans es su gran facilidad de depuración de errores y su herramienta para crear interfaces gráficas con multitud de posibilidades y todas ellas intuitivas de cara al programador. A día de hoy esta es la herramienta más fácil de manejar a la hora de crear interfaces de aplicaciones.

3. DISEÑO E IMPLEMENTACIÓN DE LA BASE DE DATOS

3.1. DISEÑO

Para la realización de la base de datos se ha tenido en cuenta un diseño de forma que se realice una asignación lo más eficiente posible y con menos problemas al trabajar con ella desde la aplicación (que se comentará en el siguiente apartado), de forma que todas y cada una de las tablas creadas tienen una función concreta para la asignación. El diseño general es el mostrado en la siguiente página:

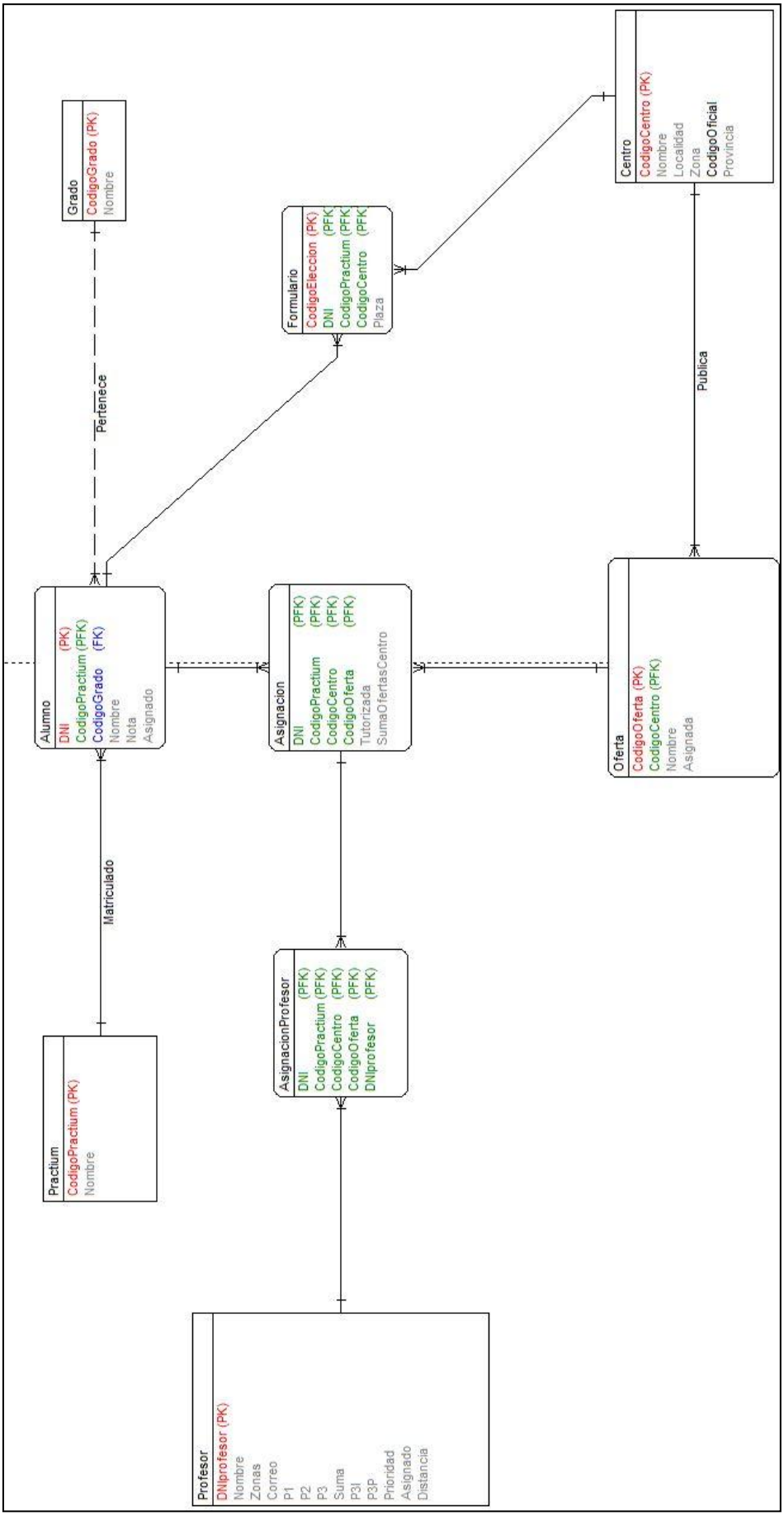


Fig. 1. Modelo entidad-relación de la base de datos.

Esta estructura guardará una muy estrecha relación con el conjunto de archivos que se recibirán en formato Excel comentados en los apartados anteriores, y con la mejor forma de gestionarlos posible.

En cada tabla se indica en color ROJO las claves primarias y únicas de cada una de las tablas de la base de datos, en color VERDE las claves que al igual que las primerias identifican de manera inequívoca dicha tabla pero que no son de la propia tabla sino que proceden de la clave primaria de otra entidad como sucede con CodigoPractium entre las clases "Practicum" y "Centro". En color AZUL se indica la clave primaria de la tabla con la que está referenciada, pero que no identifica inequívocamente a dicha tabla, sino que solo se utilizan como referencia para saber el código de la tabla adjunta (como por ejemplo entre las clases "Alumno" y "Grado" con "CodigoGrado"). El resto de valores que aparecen en tono GRIS, corresponden a variables que tiene cada una de las clases acerca de información de dicha tabla, como por ejemplo la calificación, o el código oficial de cada centro.

En el diagrama puede comprobarse también que las relaciones entre tablas formadas por líneas discontinuas son relaciones no identificativas, y por tanto en la tabla destino la clave primaria aparecerá de color Azul, mientras que si la línea es continua la clave primaria aparecerá en la tabla destino de color verde, por la razón explicada anteriormente.

En un principio, la tabla formulario no sería necesaria ya que solo almacena la información procedente de los archivos de las solicitudes de los alumnos (con las X elecciones), y esa información ya la tenemos en el archivo Excel. La razón por tanto por la que se creó esta tabla es porque a la hora de realizar la asignación desde la aplicación en Java que va a trabajar con la base de datos es mucho menos complicado de cara a triangular alumnos y ofertas si las elecciones de los alumnos ya las tenemos guardadas en la base de datos. Así pues, esta tabla no aporta información relevante sino que su utilidad radica en hacer más eficiente la asignación.

Como puede observarse en el diagrama los elementos centrales son las tablas Alumno, Oferta y Profesor, quedando las demás en un segundo plano y totalmente dependientes de la información de estas tres primeras.

Una vez realizada la asignación entre los alumnos matriculados y una de las ofertas de las X que solicitaron en su archivo de solicitudes, la aplicación

guardará cada asignación en la tabla "Asignación" como puede verse en el diagrama. Así pues, la tabla Asignación es el resultado de la primera parte del objetivo del programa que es la asignación "**ALUMNO-OFERTA**". La segunda parte sería asignar cada profesor a cada una de las tuplas de la tabla "Asignación", es decir, conseguir la unión entre "**ASIGNACIÓN-PROFESOR**".

En esta segunda parte, a medida que la aplicación vaya realizando asignaciones entre la tabla "Asignación" y la tabla "Profesor", se irán creando tuplas en la tabla "Asignación Profesor" donde cada profesor tendrá N tuplas (una por cada Practicum que se ofreció a tutorizar). Así pues, a la hora de realizar de realizar este proceso, se mirará en el Excel de los departamentos de profesores el número de PI, PII y PIII, y se buscará en la tabla "Asignación", cada tupla que tenga el código de Practicum correspondiente al que quiere tutorizar el profesor, y se irán asignando. (Más adelante se describe detalladamente)

A continuación se describen con detalle cada una de las tablas que forman parte de la estructura de la base de datos.

3.2. TABLAS

3.2.1. ALUMNO

Esta tabla nos permitirá almacenar información acerca de cada Alumno procedente de cada uno de los 24 archivos que comentamos anteriormente.

Cada alumno tendrá un DNI (clave primaria) que identificará inequívocamente a cada alumno dentro de la base de datos (solo se tendrá en cuenta la parte numérica para evitar problemas con los pasaportes, NIE de extranjeros, etc. a la hora de realizar la gestión).

Fig. 2. Tabla Alumno

Así mismo, también tiene otros atributos como el nombre que a la hora de gestionar la base de datos no sirve absolutamente de nada, simplemente es una cadena de caracteres que de cara a presentar los resultados de la asignación. El hecho de que no tenga relevancia a la hora de gestionar la información es porque para identificar inequívocamente a cada alumno ya utilizamos el DNI, que es 100% seguro que no haya dos exactamente iguales, mientras que con el Nombre si puede suceder que dos alumnos tengan exactamente el mismo nombre.

En el atributo nota se almacenará la nota media del alumno matriculado. Este campo, a diferencia del nombre, no es simplemente informativo sino que nos va a servir para ordenar los alumnos en orden descendente respecto a la nota media (primero los de PIII, luego los de PI, y a continuación los de PII), para realizar la asignación ALUMNO-FORMULARIO, y crear para cada una de esas asignaciones una nueva tupla en la tabla "ASIGNACIÓN".

Otro atributo importante a comentar es "Asignado". Como dijimos anteriormente, se creará una asignación para cada alumno y oferta en la tabla "Asignación", y este atributo sirve únicamente para saber si un alumno está ya asignado (Asignado='S') para no tenerlo en cuenta al ejecutar el algoritmo, o si no lo está (Asignado='N') para incluirlo en el proceso de reparto de alumnos.

La razón de ser de este atributo es la posibilidad por parte del usuario de introducir manualmente alumnos a la aplicación antes y después del algoritmo, de forma que es necesario distinguir a los no asignados de los que ya lo están. Así pues, al empezar la ejecución del programa todos los alumnos estarán como Asignado='N', y a medida que se vayan asignando irán cambiando su valor a 'S'.

Respecto a los dos atributos que quedan, cabe prestar especial atención al siguiente diagrama, puesto que con él se entenderá la razón de su existencia:

Fig. 3. Relación entre tablas Practicum, alumno y Grado.

Como puede observarse, la relación "Matriculado" es una relación IDENTIFICATIVA, esto quiere decir que cada alumno se va a identificar necesariamente con un DNI y con un código de Practicum (esto se debe a que cada DNI solo puede tener un Practicum, de forma que así evitamos futuros errores en la aplicación porque haya almacenado en la base de datos un alumno que esté en dos o más Practicum, algo imposible y que con esta relación evitamos que se inserten más de un Practicum para un mismo alumno). Un Practicum puede pertenecer a 1 o más alumnos pero un alumno solo podrá tener un Practicum.

Así mismo, también puede observarse la relación "Pertenece". Al contrario que la anterior, ahora se trata de una relación NO IDENTIFICATIVA, es decir, que la única diferencia radica en que para buscar a un alumno no nos fijamos en el Código del grado, solo en el DNI y en el código del Practicum, permaneciendo el código del grado solo en modo informativo. Al igual que en el anterior caso, un grado puede pertenecer a varios alumnos, pero cada alumno solo y únicamente puede estar matriculado en un grado.

La apariencia que presentará esta tabla en phpMyAdmin será la misma que en Toad Data Modeler, pero en otro formato, quedando así:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	DNI	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
2	CodigoPractium	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
3	CodigoGrado	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
4	Nombre	char(200)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único
5	Nota	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único

Como puede observarse en la anterior captura todos los atributos tendrán una longitud de 20 caracteres, a excepción del Nombre del Alumno, que se ha establecido de 200 para no tener problemas con los nombres largos a la hora de insertarlos.

3.2.2. GRADO

Aunque ya hemos comentado esta tabla en términos generales a la hora de desarrollar la tabla Alumno, vamos a entrar a explicarla más detenidamente aunque solo tiene un atributo que no hayamos comentado. Cada grado tendrá por tanto dos valores que son el código del grado (clave primaria y única para cada entrada en esta tabla), y el nombre del grado, que sucede lo mismo que con el nombre del alumno, en el sentido de que es un atributo que para realizar la asignación no influye absolutamente en nada, sino que es almacenado de modo informativo para que a la hora de obtener los resultados, estos se entiendan de forma más clara.

Fig. 4.Tabla Grado

Esta tabla no tiene por tanto ninguna complejidad a la hora de ser entendida por el lector. El aspecto de cara a phpMyAdmin quedará en este formato:

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	CodigoGrado	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
☐	2	Nombre	char(100)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único

En él puede verse que al igual que sucede con el nombre del Alumno, la longitud es de 100 caracteres, para evitar problemas a la hora de insertar determinados nombres de grados en un futuro.

En el presente proyecto, la tabla grado solo almacenará 3 tuplas que serán:

- **G420**: Grado de Magisterio en Educación Infantil.
- **G430**: Grado de Magisterio en Educación Primaria.
- **G421**: Doble Grado de Magisterio en Educación Infantil-Educación Primaria.

3.2.3. PRACTICUM

En esta tabla se almacenará la información acerca de los Practicums en los que están matriculados cada uno de los alumnos. Esta tabla presentará dos campos, en primer lugar el Código del Practicum, que es inequívoco, y que identifica de forma única cada una de las entradas, y en segundo lugar el nombre, que al igual que en el caso del Grado y del alumno, no influyen internamente para realizar la asignación no tiene ninguna utilidad, sino que únicamente se almacena con la intención de servir de modo informativo cuando sean imprimidos los resultados de la asignación.

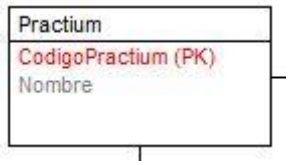


Fig. 5. Tabla Practicum.

Cabe destacar que esta tabla de cara a la asignación, solo nos es útil para saber a que Practicum pertenece cada alumno, puesto que como hemos mencionado anteriormente, los primeros en seleccionar oferta serán los del Practicum III, acto seguido los del Practicum I, y después los del Practicum II, todos ellos ordenados de mayor a menor nota media.

Al igual que la tabla del Grado, esta tabla tampoco tiene ninguna especial complejidad para el lector. El aspecto que tendrá en phpMyAdmin será el siguiente:

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
CodigoPractium	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
Nombre	char(200)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria

En la anterior captura puede verse que al igual que sucede con el nombre del Alumno y con el nombre del grado, la longitud es de 200 caracteres, para evitar problemas a la hora de insertar determinados nombres de practicums en un futuro.

En el presente proyecto, la tabla Practicum debería tener 8 (tipos de Practicum) * 3 (tipos de Grado) = 24 entradas, aunque se ha dotado a la aplicación

de escalabilidad para no tener problemas a la hora de insertar un mayor o menor número de Practicums en un futuro. Para conseguir dicha escalabilidad, el programa simplemente lee filas de alumnos, y por cada alumno nuevo que se inserta en la base de datos, se lee también el código del Practicum de ese alumno y se comprueba si ya existe en la base de datos ese código de Practicum. En caso de que ya exista, simplemente se inserta el alumno y se asocia con un código de Practicum ya existente, pero en caso de que no exista ese código de Practicum, a parte de crear el alumno, también crearemos una nueva tupla en la tabla Practicum, de forma que esas entradas pueden ser 24 como en el ejemplo descrito, o 59, puesto que depende de los datos de los alumnos que se inserten en la aplicación. Estos Practicums serían los siguientes:

Para Educación Infantil:

- **420012**: Practicum I
- **420023**: Practicum II
- **420027**: Practicum III
- **420044**: Practicum III - Conocimiento del Entorno
- **420045**: Practicum III - Lengua Extranjera
- **420046**: Practicum III - Lengua y Literatura Española
- **420059**: Practicum III – Expresión Artística Integral
- **420063**: Practicum III – Necesidades educativas especiales

Para Educación Primaria:

- **430006**: Practicum I
- **430018**: Practicum II
- **430020**: Practicum III
- **430034**: Practicum III – Las artes en educación Primaria
- **430035**: Practicum III - Lengua Extranjera
- **430049**: Practicum III – Necesidades educativas especiales
- **430051**: Practicum III – Lengua y Literatura
- **430053**: Practicum III – Educación Física

Para el Doble Grado:

- **421009**: Practicum I
- **421018**: Practicum II
- **421020**: Practicum III
- **421034**: Practicum III – Las artes en educación Primaria
- **421035**: Practicum III - Lengua Extranjera

- **421049**: Practicum III – Necesidades educativas especiales
- **421051**: Practicum III – Lengua y Literatura
- **421053**: Practicum III – Educación Física

En relación a los practicums mencionados anteriormente, hay que explicar detenidamente el **Practicum Generalista**, es decir, el Practicum III (que corresponde con los códigos **420027**, **430020** y **421020**.

En primer lugar, decir que estos códigos de Practicums no se tienen en cuenta a la hora de realizar la primera de las asignaciones ALUMNO-OFERTA, puesto que para ello tendremos en cuenta las solicitudes de los alumnos en su lista de elecciones y las ofertas disponibles en cada uno de los centros, relacionando por ejemplo un alumno que haya escogido "Maestro, educación infantil" con una plaza de Educación Infantil, pero sin tener en cuenta el código de Practicum matriculado para ese alumno. Para que la asignación sea correcta, obviamente es necesario que haya correlación entre el código matriculado por un alumno y sus elecciones, pero debe de ser el alumno el que marque las elecciones acordes con su Practicum a la hora de rellenar el formulario, y la aplicación encargada de ello la que no le permita escoger modalidades no incluidas en sus posibilidades, por lo que internamente este código de Practicum no se tiene en cuenta.

En cambio en la segunda de las asignaciones ASIGNACIÓN-PROFESOR sí que se utiliza este código, siendo esa la razón por la que se almacenan en la base de datos. En este momento ya tenemos asignado un alumno con una oferta concreta en un centro, y esa oferta a partir de ahora será identificada con un código de Practicum, que no es un atributo propio de la tabla oferta, sino que será el código de Practicum del alumno al que ha sido asignada esa oferta, es decir, en el momento que una oferta es asignada a un alumno, ésta adquiere un nuevo valor que es el código de Practicum del alumno que va a cursarla.

Esta segunda de las asignaciones consistirá por tanto en TUTORIZAR cada una de esas plazas en centros que ya han sido asignadas. Para ello, cada profesor almacenado en la base de datos tendrá un número de ofertas con PI, PII y PIII que está dispuesto a tutorizar, por lo que se trata de buscar ofertas asignadas con esos

códigos de Practicums (explicado en el párrafo anterior) e ir asignando a cada profesor a las ofertas (habrá dos métodos de elección de los profesores que los mencionaremos más adelante).

Pues bien, a la hora de realizar la asignación de los profesores, tendremos diferentes casos a la hora de buscar ofertas con códigos de Practicum:

- Ofertas que tengan **PI**:
  Estas ofertas serán aquellas que tengan los códigos 420012, 430006 y 421009 mencionadas en la lista anterior.

- Ofertas que tengan **PII:**
  Estas ofertas serán aquellas que tengan los códigos 420023, 430018 y 421018 mencionadas en la lista anterior.

- Ofertas que tengan **PIII**:
  Es aquí donde entra la explicación del **Practicum Generalista.** En la base de datos con los profesores, hay dos atributos que no hemos mencionado hasta ahora a parte del número de PI, de PII y de PIII, que son EEI (Especialidad educación infantil) y EEP (Especialidad educación primaria). Éstos pueden contener o bien dos códigos de Practicum concretos de una especialidad, o bien dos códigos que corresponden a los generalistas (420027 y 430020). Pues bien, a la hora de buscar ofertas para ser asignadas a un profesor tendremos por tanto dos casos.

  En el primero, en caso de que se haya especificado dos practicums concretos, se buscarán ofertas con esos practicums en concreto y de practicums generalistas, es decir, cualquier oferta que tenga uno de esos dos códigos concretos o uno de los dos practicums generalistas podría ser TUTORIZADA por ese profesor (4 posibilidades).

  El segundo caso es que las variables EPI y EPP no contengan códigos de practicums concretos, sino que contengan los dos códigos generalistas (420027 y 420020), por lo que en este caso no habrá 4 posibilidades de ofertas a ser TUTORIZADAS sino dos (solo aquellas ofertas que tengan o bien el código Practicum 420027 o bien 430020).

3.2.4. CENTRO

En la tabla almacenará la información acerca de los centros que han anunciado "ofertas" para Practicum, ateniendo a la información procedente de los dos archivos de los centros de los que disponemos que son el Excel de la CAM (Comunidad Autónoma de Madrid) y el de Guadalajara. Así pues, como comentaremos más adelante, la aplicación leerá estos dos archivos e irá completando esta tabla con los centros que se vaya encontrando.

Los valores almacenados serán el código del centro, el nombre, la localidad, la zona a la que pertenece, el código oficial y la provincia.

Fig. 6. Tabla Centro

El "CodigoCentro" será la clave primaria puesto que identificará de forma inequívoca a cada una de las tuplas o centros insertados. Se trata de un código que no se ha obtenido de ningún fichero de información externo, de ningún Excel, etc. (el resto de valores si), puesto que es un valor creado por la aplicación por el orden en el que se han ido insertando en la tabla, empezando en 1 y siguiendo sucesivamente hasta N centros, es decir, si hay 200 centros entre la CAM y Guadalajara, el CodigoCentro de cada uno de ellos tendrá un valor entre 1 y 200.

- ➔ A partir de ahora, a la hora de referenciar cualquier centro internamente dentro de la aplicación, el valor utilizado será este código, y no el código oficial que comentaremos a continuación.

También se nos presenta otro tipo de atributos como el Nombre, la Localidad y la provincia, que al igual que el nombre del Practicum, nombre del Grado, nombre del alumno, etc. no tiene ninguna utilidad añadida a la hora de la asignación por la aplicación, sino que su única utilidad radica en almacenar

información de modo que al obtener el resultado de la asignación final centro-alumno-profesor en formato Excel, la información acerca del centro que ofertó la plaza en cuestión aparezca más completa de cara al usuario.

Otro valor a comentar es el de la Zona. La comunidad de Madrid y Guadalajara, ha sido dividida en diferentes áreas de la forma más eficiente posible, para que la asignación del programa sea más precisa e intente asignar a profesores y a centros que estén unos cerca de los otros. La zona es por tanto un valor numérico que se nos proporciona en los archivos Excel de los centros de la CAM y de Guadalajara y que está presente para cada uno de los centros con los que el programa va a trabajar en la asignación e indica el número de área al que pertenece dicho centro. Así pues, cada oferta pertenecerá únicamente a una zona mientras que cada profesor pondrá una lista de preferencia de las zonas en las que desea ser asignado, de forma que al realizar la asignación de los profesores, se busquen ofertas que estén en alguna de las zonas que el profesor indicó en su lista de zonas de preferencia. El resultado de esta forma de asignación es que los primeros profesores en elegir seguramente sean asignados en alguna de las zonas que indicó en la lista de preferencia, mientras que los últimos es muy probable que tengan que trasladarse a zonas que no deseaban por lejanía u otros motivos.

El último de los valores a comentar de la tabla es el Código Oficial. Como puede verse en la imagen anterior, no aparece en un tono grisáceo como el resto de atributos (los que no son claves), sino que aparece en un tono oscuro. Esto se debe a que aunque NO es clave primaria, puesto que para ello ya está el código centro con un valor de 0 a N centros (como se dijo anteriormente), SI se trata de un valor ÚNICO y que NO puede ser NULO. Estas dos características (valor único y no nulo) son precisamente las características esenciales que definen una clave primaria en una tabla, por lo que el CÓDIGO OFICIAL de cada uno de los centros podría haber sido empleado como la clave primaria de esta tabla sin ningún tipo de problema y el resultado sería exactamente el mismo. La única razón por la que se ha elegido para referenciar esta tabla un numero de a 200 en lugar del código oficial (de 8 cifras normalmente), es la simplicidad de tratar con valores numéricos muchísimo más bajos.

Así pues, la única razón de ser de este Código Oficial es que los alumnos a la hora de marcar las X elecciones por orden de preferencia con la oferta que quieren y el centro que la ha publicado, el centro se indica mediante dicho código de 8 cifras, por lo que es esta la única manera de relacionar un centro con la elección de los alumnos.

Es decir, como conclusión destacar que el Código Oficial solo se utilizará única y exclusivamente para saber a qué centro se han referido los alumnos a la hora de elegir, pero el resto de las operaciones que tengan que hacerse con la tabla centro, se realizarán con el atributo Código Centro.

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
CodigoCentro	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
Nombre	char(100)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único
Localidad	char(100)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único
Zona	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único
CodigoOficial	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
Provincia	char(100)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único

Fig. 7. Tabla Centro en phpMyAdmin.

Como puede verse en la imagen anterior, que representa el formato de la entidad “Centro” en phpMyAdmin, el nombre, la localidad y la provincia tendrán una longitud de 100 caracteres por la misma razón que en las anteriores tablas (evitar problemas a la hora de introducir cadenas de caracteres demasiado largas.

3.2.5. OFERTA

Esta entidad está íntimamente relacionada con la entidad Centro. Esto se debe a que cuando leemos los ficheros de la CAM y de Guadalajara, no solo insertamos tuplas en la entidad "Centro", sino que también insertamos tuplas en esta entidad (una tupla por cada una de las ofertas que realice un centro determinado).

Así pues, si por ejemplo de la CAM y de Guadalajara recibiésemos un total de 45 centros y cada uno publicase 10 ofertas, tendríamos un total de:

- 45 tuplas en la entidad CENTRO.
- 45 * 10 = 450 tuplas en la entidad OFERTA.

Fig. 8. Tabla Oferta.

En esta entidad, como puede verse en la anterior imagen, hay un total de 4 atributos. El primero de ellos es Código Oferta, y al igual que sucede con el Código Centro comentado en el apartado anterior, no hay ningún fichero Excel o entrada externa que nos proporcione esta información, sino que su valor es un número entre 1 y el número de tuplas totales (si por ejemplo hay 450 ofertas, este valor valdrá entre 1 y 450), y será 1 de los 2 valores que servirán para identificar de forma inequívoca cualquier oferta dentro de la base de datos. Este valor numérico correspondiente a la clave primaria irá creciendo a medida que se vayan insertando ofertas, y no se ha establecido un límite (el único límite que tiene es que debe de ser un número con menos de 21 cifras, es decir, una cifra prácticamente inmensa) de forma que la aplicación es totalmente escalable a la hora de almacenar ofertas.

El segundo valor que identificará de forma inequívoca junto con el código oferta, será el código centro que ha publicado dicha oferta:

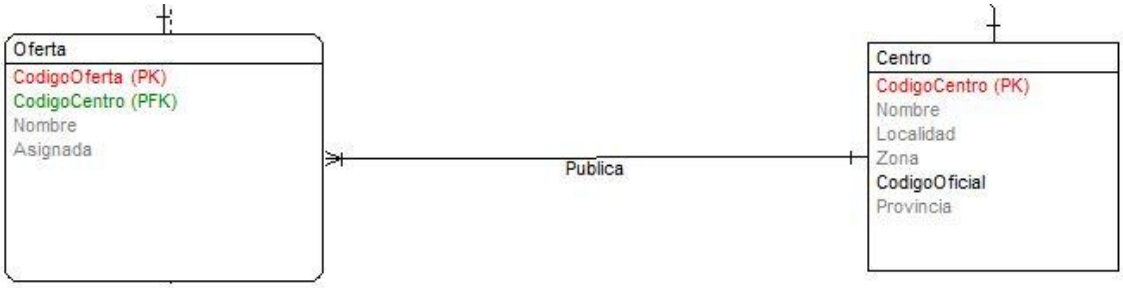


Fig. 9. Relación entre las tablas Oferta y Centro.

Así pues, como se explicó en la introducción, "Publica" se trata de una relación IDENTIFICATIVA, esto es, el código centro pasará como clave primaria a la entidad oferta, de forma que cada una de las tuplas de la tabla Oferta se identificará por el PAR de claves Código Oferta-CodigoCentro.

Esto implica que un Centro puede tener de 1 a N ofertas, pero que sin embargo cada oferta única y exclusivamente puede pertenecer a un centro.

El tercero de los atributos se referirá al Nombre de la oferta realizada por el centro. Los distintos tipos de valores que puede contener esta variable deben de ser exactamente IDÉNTICOS (la aplicación se encargará de gestionar esto unificando el formato) que los correspondientes a la tabla "Formulario" (que representa cada una de las elecciones por orden de preferencia de los alumnos), ya que la aplicación comparará ambos valores y solo asignará a un determinado alumno a una oferta si se corresponde con la elección que el solicitó en su lista. La gama de posibles valores que puede tener este campo son:

- Educación Primaria
- Educación Infantil
- Educación Física
- Educación Musical
- Lengua extranjera
- Pedagogía Terapéutica
- Audición y Lenguaje
- Educación Primaria – Bilingüe
- Educación Infantil – Bilingüe
- Educación Física – Bilingüe
- Educación Musical – Bilingüe.

Esto es, uno de los centros concreto puede ofertar por ejemplo 5 plazas de educación primaria, 2 de Educación Musical, y 1 de Audición y Lenguaje, por lo

que habría un total de 8 tuplas en la tabla Oferta con un determinado Código Centro igual para las ocho, pero cada una con un Código Oferta diferente.

En este caso el nombre no se trata de algo meramente informativo, sino que como ya hemos mencionado, es de vital importancia para poder comparar con la tabla "Formulario" que se explicará más adelante.

El último de los 4 valores corresponde a Asignada. Este valor solo puede tomar dos posibles valores en forma de cadena de caracteres que son 'S' o 'N'.

- **'N':** Valor por defecto de todas las ofertas al insertarse en la base de datos, esto quiere decir que esa determinada oferta está sin asignar, es decir, NINGÚN ALUMNO la ha elegido, por lo que está **LIBRE**.

A medida que el programa se va ejecutando, y que los alumnos van siendo asignados con el conjunto de las ofertas introducidas, se van añadiendo tuplas a la tabla "ASIGNACIÓN" (1 entrada por cada asignación ALUMNO-OFERTA realizada), que veremos más adelante.

- **'S':** Una vez que se ha creado una entrada en la tabla Asignación para una oferta y un alumno, esa oferta en cuestión es marcada con la letra 'S' ya que ahora ya está **OCUPADA** y no puede ser tenida en cuenta para el resto de asignaciones.

El objetivo por tanto del valor Asignada, es que internamente la aplicación distinga entre las LIBRES y las OCUPADAS, para solo tener en cuenta las libres a la hora de realizar el algoritmo de asignación. En realidad, este atributo no es necesario para saber si una determinada oferta ha sido ya escogida o no, puesto que en la tabla Asignación aparece el código de la oferta seleccionada por el alumno, por lo que podría comparar ese código con cada uno de los códigos de la tabla oferta y sacar el veredicto de que OFERTAS están ocupadas y cuales están libres. Sin embargó, se comprobó a la hora de acceder a dicha información, que era mucho más fácil y eficiente añadir un FLAG de 'S' o 'N' directamente en la tabla Oferta,

ya que así se evita acceder a tablas de forma innecesaria y hacer consultas complicadas que puedan dar lugar a error.

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
CodigoOferta	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
CodigoCentro	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
Nombre	char(200)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
Asignada	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria

Fig. 10. Tabla Oferta en phpMyAdmin.

En la anterior imagen puede comprobarse la estructura de dicha tabla en phpMyAdmin. En ella se ve claramente que la clave primaria la conforma el par códigoOferta y CodigoCentro, y que el Nombre de la plaza tiene también una longitud de 200 caracteres como máximo para evitar posibles errores de longitud.

3.2.6. FORMULARIO

El objetivo de la presente tabla es almacenar los datos leídos de las X elecciones de los alumnos dentro de la base de datos, con el fin de que el programa de asignación no tenga que leer más de una vez el fichero Excel, puesto que dicha información ya se queda almacenada en el sistema.

Para identificar cada elemento de la tabla utilizaremos una clave primaria formada por 4 valores (CodigoEleccion + DNI + CodigoPractium + CodigoCentro).

Pero de esos 4 atributos que forman una clave CUATRIPARTITA, solo uno de ellos es propio de la tabla, que es CodigoEleccion.

Fig. 11. Tabla Formulario

CodigoEleccion representa un numero de 1 a X elecciones realizadas por el alumno, es decir, si un alumno ha rellenado en su solicitud 8 posibles destinos, para ese alumno habrá 8 entradas en esta tabla, una con CodigoEleccion=1, otra con CodigoEleccion=2,…, y una última con CodigoEleccion=8. De forma que esta variable solo representa el orden de preferencia de una elección respecto de otra.

Esta variable llamada CodigoEleccion, no la crea el programa, sino que ya viene establecida en el Excel que se lee con los formularios de los alumnos y el programa únicamente se encarga de leerla y de almacenarla. Si un alumno tiene en el Excel 8 elecciones, las leerá de forma exactamente idéntica a si tiene 4, o si tiene 21, siendo el único requisito que no se repita el código de elección en ese fichero Excel (no puede haber dos elecciones con mismo orden de preferencia).

Las otras 3 claves que forman la clave CUATRIPARTITA, no son propias, sino que vienen de otras dos entidades, como puede verse en la siguiente imagen:

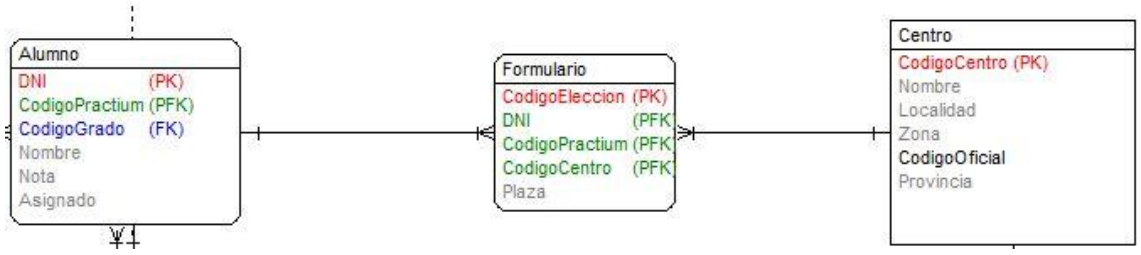


Fig. 12. Relación entre las tablas Alumno, Formulario y Centro.

Así pues, DNI y CodigoPracticum provienen de la entidad Alumno (ya que son sus claves primarias), mientras que CodigoCentro proviene del Centro, puesto que es su clave primaria. Esto es, tanto la relación que une Alumno-Formulario como la que une Formulario-Centro son relaciones IDENTIFICATIVAS, puesto que esas 3 claves que se nos han traspasado a la tabla Formulario servirán para formar la clave primaria de Formulario junto con CodigoEleccion.

Además de que esas relaciones sean identificativas, también son del orden 1:N en ambos casos, es decir, un alumno puede tener de 1 a N formularios, mientras que un formulario solo puede pertenecer a un único alumno. Lo mismo sucede entre Centro y formulario, puesto que un Centro puede tener de 1 a N formularios para hacer el Practicum es sus instalaciones, mientras que un Formulario solo puede pertenecer única y exclusivamente a un único centro.

El último atributo que nos queda por comentar es el de Plaza. Es este atributo como ya explicamos anteriormente el que nos relaciona esta entidad con la tabla OFERTA, y es el que nos referíamos antes al decir que tiene que tener uno de estos 11 valores y que debe de ser exactamente idéntico al que hay en la tabla OFERTA para que una plaza pueda ser adjudicada correctamente, aunque de eso se encargará la aplicación internamente. Los 11 posibles valores son:

- Educación Primaria
- Educación Infantil
- Educación Física
- Educación Musical
- Lengua extranjera
- Pedagogía Terapéutica
- Audición y Lenguaje
- Educación Primaria – Bilingüe
- Educación Infantil – Bilingüe
- Educación Física – Bilingüe
- Educación Musical – Bilingüe.

Es decir, a modo de explicación general, un centro publica X ofertas (tabla OFERTA), y cada oferta será de uno de estos 8 tipos obligatoriamente. Del mismo modo un alumno selecciona N posibles elecciones por orden de preferencia (tabla FORMULARIO), en el que dicho alumno indicará que CENTRO y que PLAZA quiere. A partir de aquí, la aplicación se encargará de ir comparando entre ambas tablas, mirando si para la elección de un alumno, existe una tupla OFERTA que tenga el centro y el nombre de la plaza (educación primaria, educación infantil, etc.) que el alumno desea con ese mismo nombre. Si no existe, entonces pasaríamos a la siguiente elección del alumno (elección con CodigoEleccion=2) y comprobaríamos otra vez si existe alguna tupla en OFERTA que tenga el centro indicado y la plaza indicada. Si esta segunda elección sí que existe, entonces esa oferta queda asignada, en oferta el FLAG Asignada se pone en ‘S’ para que ningún alumno pueda escogerla, y automáticamente, se crea una entrada en la TABLA ASIGNACIÓN, que indicará el alumno y su CodigoPractium, la plaza escogida y el centro seleccionado, como explicaremos en el siguiente apartado referido a la tabla ASIGNACIÓN.

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
CodigoEleccion	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
DNI	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
CodigoPractium	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
CodigoCentro	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
Plaza	char(200)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria

Fig. 13. Tabla Formulario en phpMyAdmin.

La anterior fotografía representa el formato que tendrá esta tabla en phpMyAdmin. Como puede observarse, habrá una clave CUATRIPARTITA como hemos mencionado anteriormente, y un atributo Plaza con una longitud de 200 caracteres máximo para evitar problemas futuros con la longitud.

3.2.7. ASIGNACIÓN

Continuando con el penúltimo párrafo del apartado anterior, una vez que la elección del alumno ha sido asignada (porque el centro que el seleccionó publicó la plaza que el alumno quería), el FLAG Asignada se pondrá en 'S' en la tabla OFERTA y se creará una nueva entrada en la tabla ASIGNACIÓN.

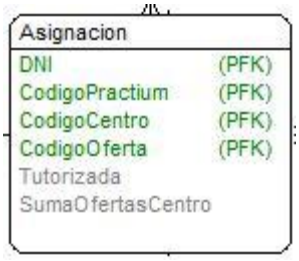


Fig. 14. Tabla Asignación

Como puede verse, esta tabla no aporta ninguna otra información adicional que no haya en otras tablas, sino que representa el resultado final de la asignación ALUMNO-OFERTA. Y es precisamente de estas dos últimas tablas de donde se obtiene la presente entidad, puesto que es fruto de una RELACIÓN N:M entre alumno y oferta como se muestra en la siguiente imagen:

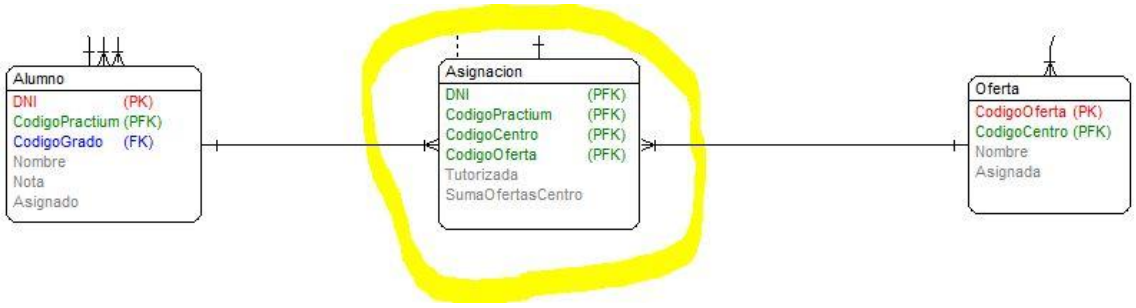


Fig. 15. Relación entre las tablas Alumno, Asignación y Oferta.

Esta tabla como puede verse, estará formada por el conjunto de las claves primarias de Alumno (DNI + CodigoPractium) y las de Oferta (CódigoOferta + CodigoCentro), puesto que al ser una relación N:M ambas relaciones son identificativas, siendo esa la razón por la cual las claves primarias pasan a la tabla asignación. También contiene otros dos atributos llamados "Tutorizada" y "SumaOfertasCentro", que explicaremos más adelante.

Cada asignación por tanto solo podrá pertenecer a un único alumno y a una única oferta. Mencionar que el CodigoCentro, como ya se explicó anteriormente, no corresponde al código oficial proporcionado por los ficheros Excel, sino que es un número de 1 a N centros. Lo mismo sucede con CódigoOferta (numero identificativo entre 1 y X ofertas). El formato en phpMyAdmin será:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	**DNI**	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
2	**CodigoPractium**	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
3	**CodigoCentro**	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
4	**CodigoOferta**	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria Único
5	**Tutorizada**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único
6	**SumaOfertasCentro**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria Único

Fig. 16. Tabla Asignación en phpMyAdmin

Esta tabla es el resultado de la finalización de la PRIMERA PARTE del proyecto, es decir, de la asignación ALUMNO-OFERTA, guardando dicho resultado en la tabla ASIGNACIÓN. Las siguientes tablas a comentar se encuadrarían dentro de la segunda parte, que corresponde a unir a un profesor con la asignación realizada, es decir, la unión PROFESOR-ASIGNACIÓN.

Será en esta segunda parte donde entre en juego el atributo "Zona" dentro de la tabla CENTRO, que no hemos utilizado para nada hasta ahora, y que lo

emplearemos para asignar a un profesor a una entrada de la tabla ASIGNACIÓN o a otra en función de sus zonas de preferencia y de donde se encuentre dicha oferta.

En cuanto a los dos atributos que faltaban por comentar, decir que "Tutorizada" representa si una oferta ya se encuentra tutorizada por un determinado profesor (flag 'S') o si no lo está (flag 'N'). La razón de ser de este atributo no es otra que la necesidad de saber que ofertas están disponibles a la hora de ejecutar el algoritmo de elección de ofertas de los profesores, y cuales se encuentran ocupadas, algo que es necesario controlar debido a la posibilidad por parte del usuario de insertar manualmente y después ejecutar el algoritmo o viceversa, todas las veces que él quiera. Cabe mencionar que este atributo no es imprescindible, en la ejecución, ya que saber que ofertas se encuentran asignadas en un determinado momento podría comprobarse restando las ofertas que están en la tabla "Asignación Profesor" (explicaremos después) y ofertas, pero de cara al diseño de os algoritmos, la creación de este atributo simplifica mucho la ejecución.

El segundo de los atributos es SumaOfertasCentro, y que es más complejo que el anterior. Este atributo tiene la utilidad de almacenar dentro de una Asignación, es decir, dentro del par "ALUMNO-OFERTA", de almacenar la cantidad de Ofertas que hay con ese mismo código de Practicum en ese centro en concreto. A modo de ejemplo, si tenemos una Asignación entre la **OFERTA** con CódigoOferta="32" y CodigoCentro="5" y un **ALUMNO** con DNI="1234" con CodigoPractium="420006", este atributo almacenará cuantas ofertas hay en la tabla Asignación con CodigoCentro="5" y CodigoPractium="420006". La utilidad reside en que a la hora de un profesor elegir los PI, PII y PIII que se ofreció a tutorizar, las ofertas sean ordenadas de mayor a menor SumaOfertasCentro, de forma que los profesores sean asignados en el menor número de centros posibles para evitar un gran número de desplazamientos. A modo de ejemplo, en la actual aplicación y con los presentes datos, un profesor que se ofreció a tutorizar un total de 24 Practicums era asignado a un total de 15 centros (porque las ofertas no eran ordenadas en base a ningún valor y su orden era aleatorio), mientras que ahora ese mismo profesor con la utilización del atributo ya mencionado es repartido en solo 3 centros, cada uno con un total de 8 ofertas.

3.2.8. PROFESOR

Esta tabla nos permitirá almacenar información acerca de cada Profesor procedente del archivo o de los archivos correspondientes a los departamentos.

Cada profesor alumno tendrá un DNI (clave primaria) que identificará inequívocamente a cada profesor dentro de la base de datos (solo se tendrá en cuenta la parte numérica para evitar futuros problemas).

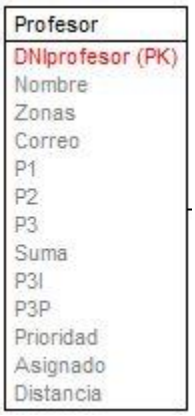


Fig. 17. Tabla Profesor

Así mismo, también tiene otros atributos como el nombre que a la hora de gestionar la base de datos no sirve absolutamente de nada, simplemente es una cadena de caracteres que de cara a presentar los resultados de la asignación. El hecho de que no tenga relevancia a la hora de gestionar la información es porque para identificar inequívocamente a cada profesor ya utilizamos el atributo DNIprofesor, que es 100% seguro que no haya dos exactamente iguales, mientras que con el Nombre si puede suceder que dos profesores tengan exactamente el mismo nombre. Lo mismo sucede con el correo, que es simplemente informativo.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	**DNIprofesor**	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
2	**Nombre**	char(200)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
3	**Zonas**	char(200)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
4	**Correo**	char(200)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
5	**P1**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
6	**P2**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
7	**P3**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
8	**Suma**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
9	**P3I**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
10	**P3P**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
11	**Prioridad**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
12	**Asignado**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria
13	**Distancia**	char(20)	latin1_swedish_ci		Sí	*NULL*			Cambiar Eliminar Primaria

Fig. 18. Tabla Profesor en phpMyAdmin.

En el atributo zonas, se almacenará una lista separada por "," de las zonas que el profesor está dispuesto a tutorizar (p ej. 1,8,14,17) y que no tiene un número concreto, sino que para cada profesor el programa puede almacenar las zonas que se deseen, aunque en el ejemplo se han puesto cuatro ya que es lo habitual.

Este atributo servirá para que a la hora de ejecutar el algoritmo se busquen tantas ofertas asignadas como ofertas dispuestas a tutorizar por el profesor, de forma que si es posible que sea repartido en los centros de alguna de esas zonas lo hará, mientras que si no es posible saldrá un aviso y se deberá proceder a la asignación manual de ese profesor para proseguir con la ejecución del programa.

Los 3 siguientes atributos merecen especial atención:

- **P1**: Indica el número de ofertas con Practicum 1 que el profesor desea tutorizar. Actualmente hay 3 posibilidades de Practicum 1 que son 420012, 430006 y 421009. Es decir, si por ejemplo un profesor tiene un valor de '7' en este atributo, ese profesor necesita tutorizar un total de siete ofertas que tengan alguno de esos 3 códigos, indistintamente, en algún o algunos de los centros de las zonas elegidas, indistintamente. (Lo ideal sería todas en el mismo centro para evitar desplazamientos, de ahí la existencia de SumaOfertasCentro).

- **P2**: Indica el número de ofertas con Practicum 2 que el profesor desea tutorizar. Actualmente hay 3 posibilidades de Practicum 2 que son 420023, 430018 y 421018. Al igual que el caso anterior, si un profesor tiene un valor de '4' en este atributo, ese profesor necesita tutorizar un total de cuatro ofertas que tengan alguno de esos tres códigos, indistintamente, en algún o algunos de los centros de las zonas elegidas, indistintamente.

- **P3**: En el comentario acerca de este atributo, aprovechamos también para explicar los valores de **P3I** y **P3P.**

  En primer lugar decir que el valor de P3 tiene el mismo funcionamiento que los dos anteriores, que es indicar el número de ofertas con Practicum 3 que el profesor desea tutorizar.

  Ahora, en diferencia con lo anterior, las posibilidades de Practicums 3 podrán ser generalistas o generalistas y específicos. Es decir, en P3I se nos almacena el código de la especialidad de INFANTIL de Practicum que ese profesor tiene conocimientos para tutorizar, y en P3P el código del Practicum concreto de PRIMARIA que ese profesor puede tutorizar. Pues bien, si por ejemplo el valor de P3='8' y P3I y P3P tienen un código de Practicum concreto, entonces ese profesor necesitará tutorizar un total de ocho ofertas que o bien tengan alguno de los códigos de Practicum especificados en P3I y en P3P, o bien Practicums generalistas (420027 y 430020), es decir, ese profesor puede tutorizar cualquier oferta que tenga uno de esos cuatro valores (dos practicums más dos practicums). Si por el contrario en P3I y en P3P los valores que aparecen son directamente los códigos de Practicums generalistas, entonces ese profesor necesitará tutorizar un total de ocho ofertas que tengan uno de esos dos valores (solamente

420027 y 430020). Su funcionamiento se explica más detalladamente en el apartado de implementación.

Para describir ahora los atributos de "Suma", "Prioridad" y "Distancia", cabe mencionar primero que a la hora de ejecutar el algoritmo de asignación de profesores con ofertas, el orden de los profesores en escoger ofertas no es aleatorio, sino que tiene dos posibles formas de establecerse:

- Algoritmo **SUMA:** En este método de ordenación de profesores, los primeros en escoger serán aquellos con un mayor número de Practicums a matricular, es decir, con el valor del atributo "**Suma**" más elevado. Este atributo almacena la suma de Practicums en total que ese profesor se ha ofrecido a tutorizar. Por ejemplo, si estos atributos tienen estos valores P1=5, P2=4 y P3=8, entonces el valor de suma será Suma=17. Así pues, el profesor con mayor "Suma" será el primero en escoger y el profesor con "Suma" más pequeño, será el último.

  Sin embargo, sucede que hay determinados valores de "Suma" para los que hay muchos profesores y es imposible establecer un método de ordenación. Por ejemplo, profesores que quieran tutorizar un total de 10 Practicum hay muchísimos. Es por ello por lo que se hace necesario incluir otro atributo, que es "**Distancia**". Este atributo indica la distancia de la primera letra del apellido del tutor respecto de la letra escogida por el usuario como criterio de ordenación a la hora de iniciar el programa. Es decir, si por ejemplo la letra establecida por el usuario fue la "C", el algoritmo primero seleccionará los profesores con mayor número de practicums en total a tutorizar, y en caso de que haya más de un profesor con ese valor, realizará dentro de los mismos a su vez otra ordenación de profesores, de forma que si el número de Practicums donde hay repetidos es 10, el primero en escoger será el profesor con 10 Practicums

cuyo apellido empiece por C, el segundo cuyo apellido empiece por D, etc., y el último será el profesor cuyo apellido empiece por B (da la vuelta al abecedario). En caso de que la "Distancia" también coincida, entonces se ordenará por orden alfabético, es decir, por la segunda letra del apellido, luego la tercera, etc. Así pues, en caso de empate la ordenación de los profesores siempre se realizará de forma alfabética en base a su apellido por lo que la única razón de ser del atributo "Distancia" es la posibilidad del usuario de elegir una letra de preferencia.

- Algoritmo **PRIORIDAD**: El funcionamiento de este algoritmo es mucho más simple que el anterior. Para su ejecución se hace uso del atributo "Prioridad". Este atributo simplemente indica un número de 1 a N profesores, que indica el orden ya establecido por el usuario (puesto que este valor a diferencia de los dos anteriores si que se indica en el fichero Excel de entrada) en el que se han de repartir los profesores. Este algoritmo decidió implementarse debido a la posibilidad de ciertos criterios de preferencia no contemplados en el programa por los que un profesor tenga que se asignado antes que otro.

El último de los atributos a mencionar es "Asignado". Este atributo existe porque se hace necesario saber internamente que profesores tienen ya la totalidad de las plazas que decidieron tutorizar asignadas, para no ser tenidos en cuenta en las posteriores ejecuciones del algoritmo, de forma que tendrá dos posibles valores que serán flag='S' (ya está asignado) y flag='N' (todavía no está asignado). Así pues, este atributo se hace indispensable ya que a la hora de ejecutar el algoritmo, si un profesor no ha podido ser asignado, saldrá un mensaje de alerta para que se proceda a su asignación manual. Pues bien, después de esa inserción manual, si se desea volver a la ejecución del algoritmo, es necesario separar a la hora de ordenar los profesores los que están ya asignados y no debemos tener en cuenta de los que no lo están.

Mencionar que obviamente al empezar todos los profesores tendrán este atributo con valor 'N', y a medida que vaya ejecutándose la aplicación, irán cambiando su valor a 'S'.

3.2.9. ASIGNACION PROFESOR

La creación de cada una de las tuplas en la presente tabla supone la terminación del proceso de la aplicación, es decir, el fin último de este programa es crear tuplas en esta entidad. Así pues, esta tabla representa el resultado final del proceso de Asignación, indicando el DNI del alumno en cuestión, la OFERTA del centro que ha escogido, y el PROFESOR que se va a encargar de tutorizar a ese alumno en esa oferta.

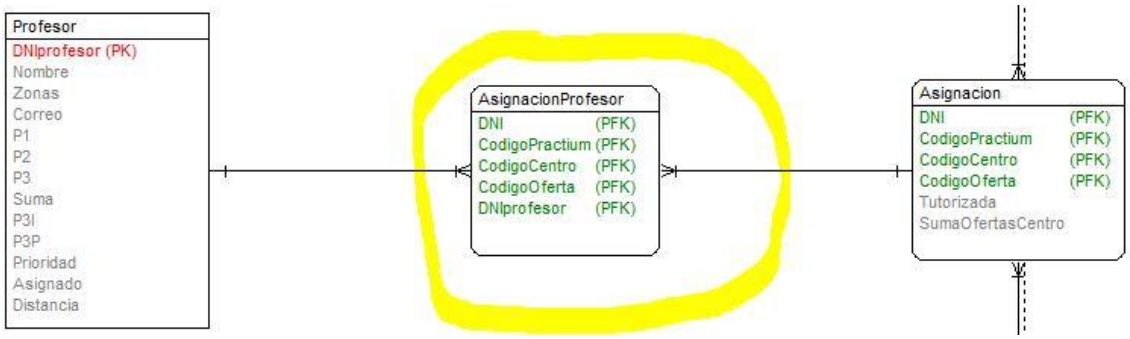


Fig. 19. Relación entre las tablas Profesor, Asignación Profesor y Asignación.

Esta tabla por tanto será una relación N:M entre Profesor y Asignación (recordemos que asignación representa el emparejamiento ALUMNO-OFERTA), y sus claves primarias serán por tanto el conjunto de claves de la entidad Profesor junto con las claves de la tabla Asignación.

Fig. 20. Tabla Asignación Profesor.

Así pues, sus valores serán el **DNI** (DNI del alumno que ha sido emparejado con la oferta en cuestión), **CodigoPractium** (El código de Practicum a la hora de matricularse de ese mismo alumno), **CodigoCentro** (El código del centro al que pertenece esa oferta), **CódigoOferta** (El código de la oferta en cuestión) y **DNIprofesor** (clave primaria de la entidad profesor representante de su DNI).

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
DNI	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
CodigoPractium	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
CodigoCentro	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
CodigoOferta	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria
DNIprofesor	char(20)	latin1_swedish_ci		No	*Ninguna*			Cambiar Eliminar Primaria

Fig. 21. Tabla Asignación Profesor en phpMyAdmin.

Cabe mencionar como es el proceso final de asignación que permite insertar tuplas en esta entidad. Pues bien, a la hora de ejecutar uno de los dos algoritmos para asignar profesores (Algoritmo de suma o algoritmo de prioridad), a medida que se vayan analizando profesores y ubicándolos en las distintas ofertas (tantas como solicitaron tutorizar en el archivo de los departamentos), ocurrirán 3 cosas:

- **Tutor asignado.**
  El tutor en cuestión que acaba de ser asignado a todas y cada una de las ofertas que ha solicitado, en distintos o en el mismo centro, será marcado con el flag Asignado=’S’,
- **Asignación tutorizada.**
  Cada una del total de las asignaciones que ahora serán tutorizadas por el profesor será marcada con el flag Tutorizada=’S’. (Recordemos que la entidad Asignación representa el emparejamiento ALUMNO-OFERTA).
- **Nueva tupla en “Asignación Profesor”.**
  Se insertará en la tabla Asignación profesor (que es la que estamos describiendo en este apartado), una nueva tupla con los datos del emparejamiento final PROFESOR-ALUMNO-OFERTA.

Así pues, si por ejemplo un Profesor solicitó tutorizar seis Practicum 1, cuatro Practicum 2, y diez Practicum 3, en total deberá tutorizar un total de 20

ofertas (6 + 4 + 10). Pues bien, a medida que se vaya ejecutando el proceso de asignación, cada una de esas 20 ofertas que vaya asignando a ese profesor (en algún centro de las zonas que eligió), le pondrá el flag Tutorizada=’S’, y creará en la entidad Asignación Profesor una nueva tupla. Al final, cuando hayan sido asignadas 20 ofertas a ese profesor, entonces ya se habrá terminado con él y se le pondrá el flag Asignado=’S’. Al final, para este profesor en concreto, tendríamos al tutor con el flag Asignado=’S’, un total de 20 nuevas tuplas en la entidad Asignación profesor, un total de 20 nuevas entradas en la tabla “Asignación” que ahora tienen el flag Tutorizada=’S.

3.3. IMPLEMENTACIÓN

3.3.1. DETALLES GENERALES

Para poder implementar esta base de datos y las entidades ya explicadas se ha utilizado el gestor de base de datos **MySQL** con interfaz gráfica de cara al usuario **phpMyAdmin**, que viene incluido dentro del paquete **XAMPP**.

En el presente apartado se indicarán a modo general los pasos seguidos para proceder a la creación de la base de datos y de las entidades que la conforman.

Para realizar el diseño de la base de datos, se recurrió al programa externo **Toad Data Modeler**, acto seguido se realizaron los siguientes pasos:

PASO 1: Diseño del diagrama

Después de diseñar la estructura de la base de datos, pulsaremos en el botón de la barra de herramientas llamado "Generate SQL" y se nos abrirá una ventana como la siguiente (Figura 22), en la que activaremos todas las casillas, indicaremos la ruta donde se nos va a guardar el Script con sentencias en SQL y pulsaremos el botón "Generate". Una vez hecho esto se nos habrá guardado un dichero en formato ".sql" en la ruta que le hayamos indicado, así que ya tenemos las instrucciones que hay que ejecutar en la nueva base de datos (que consistirán en la creación de las tablas, relaciones, permisos, etc. que hemos creado hasta ahora).

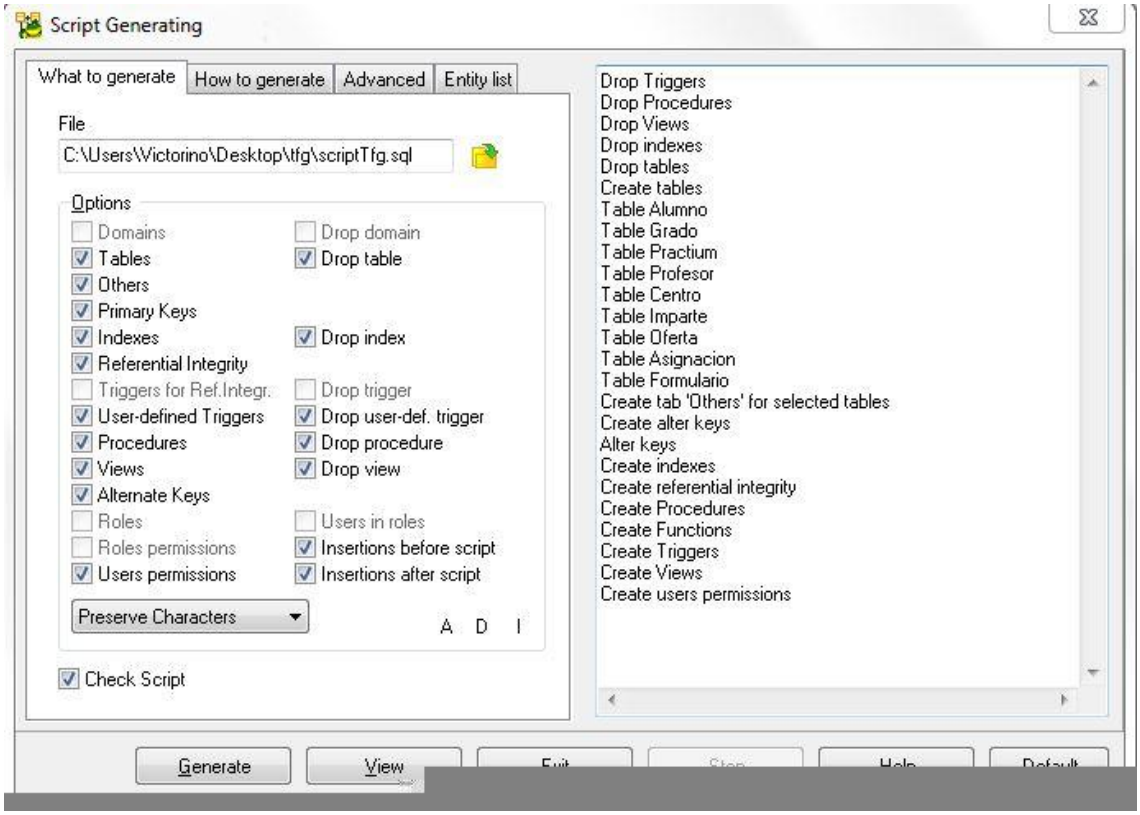


Fig. 22. Generación del archivo SQL.

PASO 2: Creación de la base de datos.

Ahora, dentro del entorno phpMyAdmin, procederemos a la creación de la base de datos, para poder ejecutar las sentencias fruto del anterior paso (y que provocarán la creación de todas las tablas, relaciones, permisos, etc.). Para ellos pulsaremos en crear nueva base de datos, y acto seguido aparecerá:

Fig. 23. Proceso de selección de nombre de nueva base de datos en phpMyAdmin.

Seleccionaremos un nombre de la nueva base de datos, y pulsaremos en Crear. En lugar de hacerlo en modo interfaz, también podríamos crear la base de datos mediante sentencia, escribiendo en la consola SQL el siguiente comando:

CREATE DATABASE nuevaBase

Ahora, una vez creada la base de datos, sería recomendable asignarle una contraseña puesto que por defecto no viene ninguna, aunque para el desarrollo del proyecto no se ha asignado ninguna. Ahora ya tenemos nuestra nueva base de datos creada, y pasaremos a lo siguiente, que es crear todas las tablas.

PASO 3: Creación de las tablas.

Ahora procederemos a crear las tablas que hemos diseñado en el paso 1, y para ello, aunque se podría ir insertando las tablas de una en una, insertando las sentencias SQL en la línea de comandos, lo que haremos será IMPORTAR el fichero que generamos en el paso 1, de forma que se ejecutarán todas las instrucciones de golpe. El formato del fichero creado será este:

```
drop table IF EXISTS Formulario;
drop table IF EXISTS Asignacion;
drop table IF EXISTS Oferta;
drop table IF EXISTS Imparte;
drop table IF EXISTS Centro;
drop table IF EXISTS Profesor;
drop table IF EXISTS Practium;
drop table IF EXISTS Grado;
drop table IF EXISTS Alumno;


Create table Alumno (
        DNI Char(20) NOT NULL,
        CodigoPractium Char(20) NOT NULL,
        CodigoGrado Char(20) NOT NULL,
        Nombre Char(200),
        Nota Char(20),
 Primary Key (DNI,CodigoPractium)) ENGINE = My

Create table Grado (
        CodigoGrado Char(20) NOT NULL,
        Nombre Char(100),
        UNIQUE (CodigoGrado),
 Primary Key (CodigoGrado)) ENGINE = MyISAM;

Create table Practium (
        CodigoPractium Char(20) NOT NULL,
        Nombre Char(200),
        UNIQUE (CodigoPractium),
```

Fig. 24. Ejemplo de formato de un fichero SQL.

Ahora importaremos este fichero en nuestra base de datos pulsando en el botón de la barra superior llamado “IMPORTAR”, y seleccionaremos el archivo SQL de esta manera:

Fig. 25. Importación de archivo SQL en phpMyAdmin.

Acto seguido, abajo pulsaremos el botón continuar, y ello implicará que se habrán ejecutado todas las sentencias de creación de tablas, por lo que habrá finalizado la creación de la estructura de nuestra base de datos, tal que así:

Fig. 26. Ejemplo de mensajes devueltos después de una consulta SQL en phpMyAdmin.

Ahora como podemos ver, ya tenemos todas las tablas de nuestra base de datos, y podemos acceder a cada una de ellas:

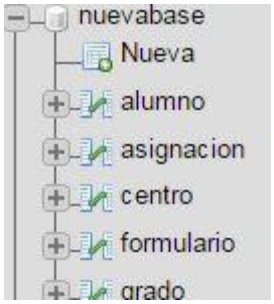


Fig. 27. Ejemplo de estructura de una base de datos en phpMyAdmin.

PASO 4: Insertar Datos.

Con la estructura de datos creada, solo nos faltaría insertar los datos en cada una de las tablas. Esto se puede hacer bien con la interfaz gráfica de phpMyAdmin, o bien mediante sentencias SQL. Nuestra opción escogida será la segunda, puesto que la creación de tuplas se realizará desde Java con nuestra aplicación a medida que se vayan leyendo los ficheros Excel de entrada y que se vayan realizando las asignaciones. Aun así, explicamos de cara al lector como se crearían las tuplas para las tablas en caso de que quisiera hacerlo por medio de la interfaz. En el siguiente ejemplo se quiere insertar en la tabla Alumno:

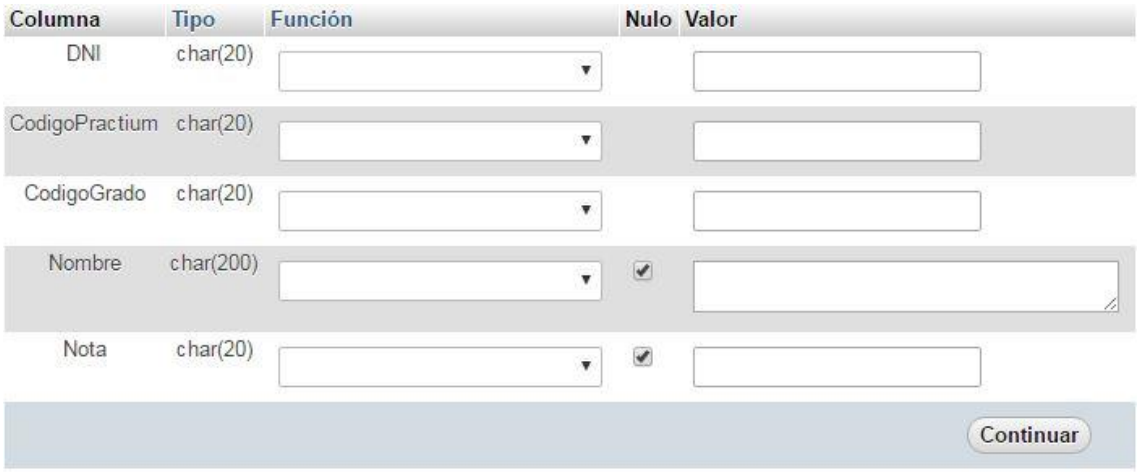


Fig. 28. Ejemplo de inserción de una tupla en una tabla por medio de la interfaz de phpMyAdmin.

Una vez creada la base de datos, procederemos a explicar el proceso que realiza la aplicación para recibir los datos, su tratamiento mediante la inserción en de tuplas en la base de datos, y los correspondientes resultados obtenidos.

4. RESULTADOS

4.1. FICHEROS RESULTANTES

Antes de pasar a comentar la implementación del código, mencionar que todos y cada uno de los resultados mencionados anteriormente serán generados en Excel y mostrados al usuario en el proceso de asignación. Tendremos por tanto los siguientes archivos:

- **Alumnos de Intercambio.**

  Lista de alumnos que a la hora de leer los archivos de las matriculas se ha descubierto que no están matriculados en uno de los 3 grados disponibles (G420, G430 y G421). No serán tenidos en cuenta para el proceso.

Archivos de la primera parte **ALUMNO-OFERTA**

- **OfertasSinAsignar.xlsx**

  Lista de ofertas con el flag Asignada=’N’, es decir, ofertas que han publicado los centros que todavía no tienen un alumno para cursarlas.

- **OfertasAsignadas.xlsx**

  Lista de ofertas con el flag Asignada=’S’, es decir, ofertas que ya se encuentran asignadas con un alumno en concreto.

- **PI_AlumnosSinAsignar.xlsx**

  Lista de alumnos matriculados en alguno de los Practicum I de alguno de los grados, y que todavía no han sido asignados a ninguna oferta, es decir, ninguna de sus posibles elecciones en el formulario se ha podido cumplir.

- **PII_AlumnosSinAsignar.xlsx**

  Mismo caso que PI_AlumnosSinAsignar.xlsx pero para los alumnos de alguno de los Practicum II.

- **PIII_AlumnosSinAsignar.xlsx**

  Mismo caso que los anteriores pero para los alumnos de alguno de los Practicum III.

Archivos de la segunda parte **PROFESOR-ASIGNACIÓN**

- **OfertasSinTutorizar.xlsx**

  Lista de ofertas presentes en la tabla asignación con el flag Tutorizada=’N’, es decir, son ofertas que tienen un alumno asignado ya para cursarlas, pero que todavía no se encuentran tutorizadas por ningún profesor.

- **ResultadoFinal.xlsx**

  Representa el resultado final de la ejecución del programa, es decir, aquella lista de ofertas que han sido asignadas a un alumno en concreto, y que además ya se encuentran tutorizadas por un profesor específico.

5. DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN

5.1. INTRODUCCIÓN

Todo lo expuesto en los apartados anteriores forma parte de la aplicación también, pero forma parte en lo que al desarrollo de la estructura de las tablas y a la organización de los datos se refiere. En el presente apartado se procederá a explicar cómo se ha desarrollado la aplicación, sus funcionalidades, como accede a los datos en la base de datos, como los inserta, como los modifica, como lee los archivos Excel con la información externa, como gestiona esos Excel de forma eficiente, etc.

Así pues, la razón por la que la parte de la base de datos ha sido separada de la aplicación es porque ahora nos centraremos en la parte de programación, explicando líneas de código y su funcionamiento, sin prestar demasiada atención a la estructura de las tablas y a las relaciones entre ellas que ya han quedado explicadas en apartados anteriores, puesto que se sobreentiende que el lector ya tiene una ligera noción de la estructura de la información.

A modo general, mencionar que la presente aplicación ha sido desarrollada con el lenguaje de programación JAVA sobre la plataforma NETBEANS por dos razones. La primera es que Netbeans ofrece una gran interfaz gráfica para programar interfaces, y la segunda es que realiza acceso a bases de datos de forma eficiente e intuitiva. Esa base de datos a la que accederemos será es MySQL y para su comprobación gráfica utilizaremos PHPMYADMIN (sobre la plataforma XAMPP), habiendo diseñado toda la estructura de tablas y relaciones antes de ello con el programa TOAD DATA MODELER.

Así pues, lo que la aplicación pretende es pedirle al usuario todos los archivos que necesita para hacer la asignación, y una vez hecho esto y comprobado que no hay error en el formato de éstos, proceder a manipular la información. Pero para manipular la información primero hay que almacenar toda esa información en algún sitio, que será en la base de datos, y cuya dirección será establecida por defecto en la clase de java llamada ModeloBBDD. Ahora, una vez leídos todos los

Excel, y haber pasado esa información a la base de datos en un formato estándar que nuestra aplicación pueda leer, se procederá a la utilización de los algoritmos que gestionen la asignación de Practicums.

Sin embargo, antes de ponerse a programar lo expuesto anteriormente, es necesario saber cuál debería ser la estructura de la aplicación teniendo en cuenta lo ya mencionado. Así pues, la aplicación recibirá Excel (Alumnos + Centros + Elecciones Alumnos + Profesores), leerá esos archivos, y llamará a la base de datos para almacenar toda esa información. Una vez que toda la información sobre centros, alumnos, etc. ha sido almacenada en la base de datos, se procederá a pasar a la parte de ASIGNACIÓN DE PRACTICUMS. Esos algoritmos de asignación de Practicums, los dividiremos en dos grandes bloques. Por un lado se encontrará la parte dedicada a asignar ALUMNOS y OFERTAS, y una vez realizada esa asignación por medio de la aplicación y haber almacenado dicho resultado en una tabla de la base de datos llamada "Asignación" como mencionábamos en el apartado anterior, procederemos a pasar a la segunda parte. La segunda parte se encarga de realizar la asignación OFERTA-PROFESOR.

Con esta ultima parte ya habremos finalizado todo el proceso de asignación y nuestro programa habrá TERMINADO, sacando en una serie de Excel (para que el usuario lo lea de forma intuitiva) no solo los resultados finales de la asignación, sino también los alumnos, profesores, ofertas, etc. que no han podido ser asignados durante la ejecución del programa. La finalidad por tanto de estos archivos no es solo mostrar la información al usuario, sino también permitirle realizar asignaciones manuales de alumnos y de profesores en caso de que el algoritmo no haya podido asignarlos correctamente. Todo lo anteriormente expuesto se muestra de forma intuitiva en el siguiente diagrama:

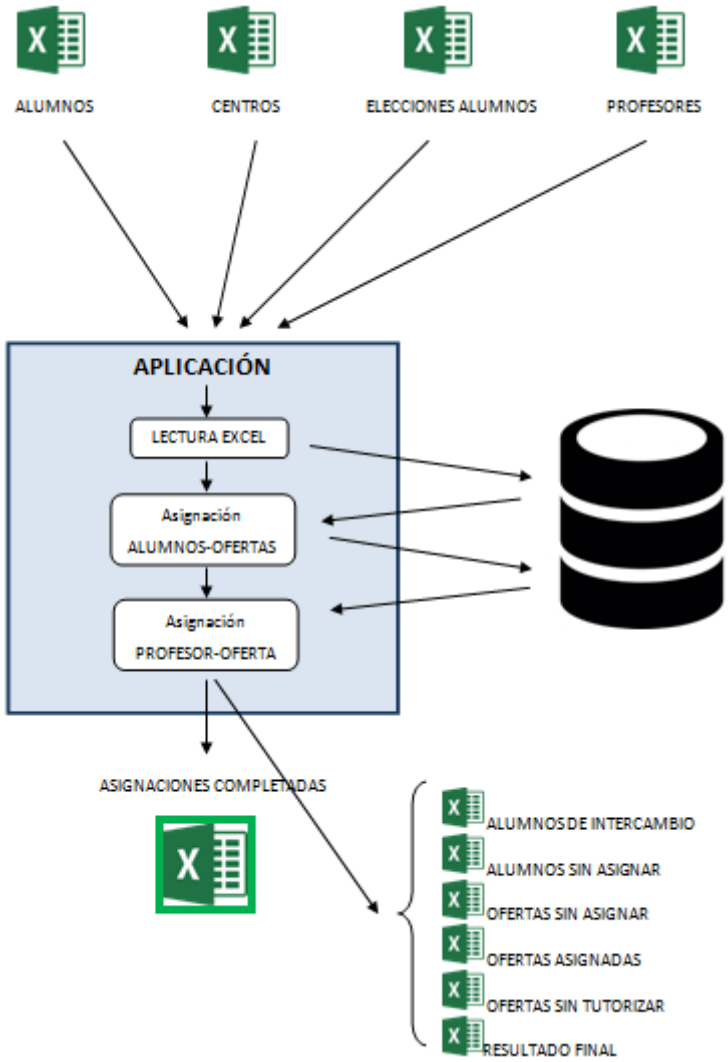


Fig. 29. Diagrama de ejecución con los ficheros de entrada y de salida implicados.

Así pues, una vez realizado el diagrama de los pasaos de ejecución que realizará el programa se procederá a explicar los procesos internos y líneas de código que el usuario estándar no verá, pero que son relevantes para entender el funcionamiento de la aplicación. Toda esa información en Java que contiene la aplicación, se divide en tres grandes paquetes (Package), que son Iconos, Interfaz, y Lógica de Negocio, y que serán explicados en los siguientes apartados.

5.2. LÓGICA DE NEGOCIO

En este paquete es donde se encuentra el BackEnd de la aplicación, es decir, donde se encuentran todas las líneas código que corresponden a la realización de los algoritmos que permiten la asignación de practicums. En este paquete se encuentra el 90% del trabajo desarrollado para la aplicación y es aquí se encuentran todos los métodos de lectura de ficheros Excel, así como los métodos encargados de intercambiar con información con la base de datos.

Es decir, esta es la parte intocable de la aplicación para que funcione correctamente. Tanto el paquete interfaz como el paquete iconos solo contienen información acerca de cómo es representada la información al usuario, forma de los botones, acciones al realizar cada uno de ellos, iconos, etc., pero que puede ser modificada en un futuro o darle otra apariencia y la aplicación seguiría funcionando perfectamente, mientras que en cambio este paquete contiene la información esencial.

Así pues, como conclusión decir que la aplicación podría funcionar perfectamente solo con lógica de negocio, sin interfaz, pero el caso inverso no podría darse nunca. Las clases que lo componen son las siguientes:

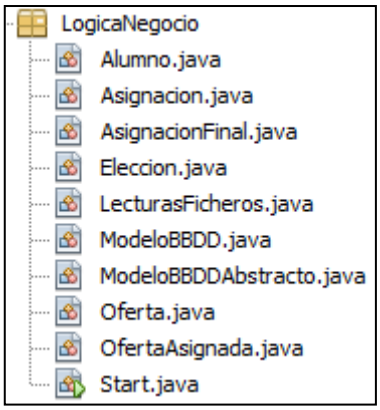


Fig. 30. Clases en Java que conforman el paquete LogicaNegocio.

5.2.1. Alumno.java

Esta entidad contiene los atributos y métodos que representan la tabla "Alumno" dentro de la base de datos:

```
public class Alumno {
    private String DNI;
    private String CodigoPractium;
    private String CodigoGrado;
    private String Nombre;
    private String Nota;
```

Fig. 31. Atributos de la clase Alumno en Java.

Esta entidad será utilizada dentro de las líneas de código para dividir a los alumnos en ArrayList de forma que se puedan separar los que estén matriculados en PI, PII o PIII, o para agrupar los alumnos que se han quedado sin asignar.

5.2.2. LecturasFicheros.java

En esta clase es donde se agrupan todos los métodos que se encargan de la lectura de ficheros Excel por parte de la aplicación. Aunque dentro del código de cara al programador se encuentran detalladas las especificaciones que realiza cada función, las comentaremos aquí de forma general. En primer lugar comentaremos las cuatro grandes funciones que hay en esta clase (una por cada icono Excel que se aprecia en el diagrama general mostrado en el apartado anterior), y a continuación los métodos que sirven de apoyo a esas funciones para hacer que sea posible este proceso.

En primer lugar se realiza la lectura de los ficheros que contienen información acerca de los **CENTROS** y de las **OFERTAS** que estos han publicado, y que se divide a su vez en dos partes:

- **GUADALAJARA**

El método que se encargará de la lectura de un único Excel y que se corresponde con los centros de Guadalajara es este:

```
public static void leerCentrosGuadalajara(ModeloBBDD mod) throws IOException {
```

Este método recibirá la dirección del archivo donde se encuentran los centros de Guadalajara, que tendrá este formato:

LOCALIDAD	CENTRO	ESPECIALIDAD										ZONA	
	NOMBRE DE CENTRO	EP	EI	EF	MÚ	ING	PT	AL	EPB	EIB	EFB	MÚB	Código centro

Fig. 32. Formato del fichero Excel de entrada de los centros de Guadalajara.

Y se encargará de leer línea por línea todos y cada uno de los centros, así como el número y el tipo de ofertas que publica cada uno, yendo a crearCentro() dentro de la clase ModeloBBDDAbstracto por cada uno de los centros que lee correctamente. El método terminará cuando ya no haya más líneas en el fichero Excel para leer.

- **MADRID**

El método que se encargará de la lectura de un único Excel y que se corresponde con los centros madrileños es este:

```
public static void leerCentrosMadrid(ModeloBBDD mod) throws IOException {
```

El método, al igual que sucede con el que recibe los centros de Guadalajara, recibirá la dirección del fichero donde se encuentra la información, y cuyo formato no será tan simple como el de Guadalajara, puesto que contiene muchísimas mas información. Además, mientras que en Guadalajara cada línea corresponde a un centro y en esa misma línea se indican todas las ofertas que ese centro publica, en Madrid cada línea indica una oferta, es decir, un mismo centro puede tener de 1 a N líneas.

Así pues, una vez realizada la lectura de los dos archivos correspondientes a los centros, pasaremos al segundo de los tipos de Excel que hay que leer. Este segundo Excel a leer será el correspondiente a los **ALUMNOS**. A diferencia de los centros, no

se trata solo de un archivo de la comunidad de Madrid y otro de Guadalajara, sino que en este caso se trata de una amplia lista de archivos en formato CSV, concretamente un archivo por Practicum y Grado, es decir, si el usuario metiese la totalidad de archivos que debería en este caso, serían 8 (Practicum) * 3 (Grados) = 24 archivos en formato .csv.

Estos archivos contendrán información de cada alumno (un alumno por fila) como su DNI, su Nombre, su nota media, el grado al que pertenecen, y el Practicum al que se han matriculado. El método que se encargará de su lectura será el siguiente:

```
public static void leerMatriculas(ModeloBBDD mod, String ruta) throws IOException {
```

Al igual que sucede con los métodos de leer centros anteriores, recibirá la ruta del archivo que contenga los alumnos que desea leer, por lo que en caso que tuviésemos 24 archivos correspondientes a las matrículas de los alumnos, habría que llamar a este método 24 veces (una por fichero).

Dentro del método, se llamará a tres métodos que se encuentran dentro de la clase ModeloBBDDAbstracto:

- crearPractium(): Insertará una tupla en la tabla Practicum que vimos en el diseño de BBDD comentado unas páginas atrás. Al finalizar el proceso debería haber un total de 24 tuplas.
- crearGrado(): Inserta una tupla en la tabla Grado que vimos en el diseño de BBDD comentado anteriormente. Al finalizar el método debería haber creado un total de 3 tuplas en dicha tabla.
- crearAlumno(): Llamará a este método una vez por cada alumno que se encuentre dentro del archivo (un alumno por línea), pasándole todos los datos como DNI, Nota, etc.

Así mismo, los alumnos que se encuentre en la lectura de estos ficheros que no tengan nota media, o que sean de Intercambio (se les identifica porque su código de grado es diferente), se les va añadiendo un Array llamado alumnosSinAsignar, y al finalizar la ejecución de la función se generará un Excel con la lista de alumnos de intercambio que nos hemos ido encontrando en la lectura, y que no van a formar parte del proceso de asignación de nuestro programa.

En tercer lugar, tenemos la lectura de las elecciones de los alumnos, o lo que es lo mismo, los **FORMULARIOS**. Esta lectura tiene la misma característica de la lectura de las matrículas de los alumnos en el sentido de que no se trata de un único archivo, sino que son una gran cantidad de ficheros. El método que se encarga de esta lectura es el siguiente:

```
public static void leerFormulario(ModeloBBDD mod, String direccion) throws IOException {
```

En estos ficheros, cada línea se corresponde con una elección de un alumno, con un numero de elección, por lo que cada alumno tendrá de 1 a N líneas, una por cada número de elección (están ordenadas por orden de preferencia del alumno, siendo la elección 1 la preferible a seleccionar). Este método recibirá también la ruta del archivo a leer, y leerá únicamente un archivo, por lo que será llamado un total de X veces, una por fichero.

Una vez dentro del método, leerá la línea correspondiente a la elección del alumno, y en dicha línea se indicará la plaza que el alumno desea, siendo siempre una de este abanico de opciones:

```
//Normales
if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Infantil"))
    Plaza="Educacion infantil";
else if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Especial"))
    Plaza="Educacion especial";
 else if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Primaria"))
    Plaza="Educacion primaria";
 else if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Fisica"))
    Plaza="Educacion fisica";
else if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Musical"))
    Plaza="Educacion musical";
else if (palabrasEnLinea[6].equals("Maestro, especialidad Lengua Extranjera"))
    Plaza="Lengua extranjera";


//Bilingues
if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Infantil - B"))
    Plaza="Educacion infantil - B";
else if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Especial - B"))
    Plaza="Educacion especial - B";
 else if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Primaria - B"))
    Plaza="Educacion primaria - B";
 else if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Fisica - B"))
    Plaza="Educacion fisica - B";
else if (palabrasEnLinea[6].equals("Maestro, especialidad Educacion Musical - B"))
    Plaza="Educacion musical - B";
else if (palabrasEnLinea[6].equals("Maestro, especialidad Lengua Extranjera - B"))
    Plaza="Lengua extranjera - B";
```

Fig. 33. Lista de posibles elecciones que pueden hacer los alumnos al comenzar el curso.

Como puede verse en la anterior imagen, cada vez que leemos alguno de los tipos anteriores, el método se encargará de cambiar dicho nombre (según el tipo que sea), por uno estandarizado dentro de la base de datos, de forma que a la hora de emparejar la tabla oferta y formulario haya correlación entre ambas, con la característica peculiar de que los alumnos que hayan escogido en su elección Educación especial, serán asignados en Pedagogía terapéutica o en Audición y Lenguaje de forma totalmente aleatoria en el centro que hayan escogido, ya que ambas plazas pueden ser ocupadas por este tipo de alumnos.

Así pues, una vez leído, se llamara a crearFormulario() indicando el número de la elección el DNI del alumno que la solicitó, el código del centro donde quiere cursar dicho praticum (Código Oficial), y la Plaza a solicitar (uno de los 10 nombres ya descritos, en formato estandarizado).

En cuarto y último lugar, tenemos la lectura de los archivos correspondientes a las elecciones de los **PROFESORES**. Esta aunque en el presente proyecto se realiza por medio de un único archivo, se ha dotado a la aplicación de la capacidad de leer un número ilimitado de archivos de profesores para no tener problemas con la escalabilidad. El método que se encarga internamente de la lectura es el siguiente:

```
public static void leerProfesores(ModeloBBDD mod, String direccion)
```

Así pues, esta función recibirá la dirección del archivo a leer e irá insertando uno por uno los profesores que se vaya encontrando.

Cabe mencionar que Todos los DNI con que lea la aplicación por medio de ficheros, excluirá los caracteres alfabéticos y se quedará solamente con los caracteres numéricos. Es decir, los DNI de Alumnos y de Profesores están formados internamente solamente por números. Así mismo, en correlación con los 4 métodos de lectura comentados anteriormente, hay 2 métodos más que sirven de apoyo a estos, que son:

```
public static String buscarZona(String[] palabrasEnLinea) throws IOException {
    for (int i=0; i<palabrasEnLinea.length; i++)
        if (palabrasEnLinea[i].length()==2 && Integer.parseInt(palabrasEnLinea[i])>=15)
            return palabrasEnLinea[i];
    return "0";
}
```

Fig. 34. Código de la función buscar Zona.

Este primero se encarga de recibir todos los atributos de un centro concreto de Guadalajara, y devolver el atributo correspondiente al que indica la zona a la que pertenece, indicando su valor.

```
public static String eliminarAcentos(String str) {

    final String ORIGINAL = "ÁáÉéÍíÓóÚúÑñÜü";
    final String REEMPLAZO = "AaEeIiOoUuNnUu";

    if (str == null) {
        return null;
    }
    char[] array = str.toCharArray();
    for (int indice = 0; indice < array.length; indice++) {
        int pos = ORIGINAL.indexOf(array[indice]);
        if (pos > -1) {
            array[indice] = REEMPLAZO.charAt(pos);
        }
    }
    return new String(array);
}
```

Fig. 35. Código de la función eliminar Acentos.

Este segundo método en cambio se encarga de recibir una cadena de caracteres y devolverla habiendo eliminado todos los acentos y caracteres especiales que dicha frase pudiera contener. Este método se creó porque a la hora de subir datos a la base de datos de cualquiera de las tablas, si alguno de los campos contenía acentos o caracteres especiales, ese carácter era eliminado, de forma que para que la información almacenada vaya en correlación es necesario llamar a esta función cada vez que creamos alguna tupla (la que sea) dentro de la base de datos.

Existirán dos métodos también complementarios al resto que se encargarán de la inserción manual. En primer lugar tenemos **leerAlumnoOfertaManual** , que

recibirá como parámetro la dirección de un archivo en formato CSV en el que el usuario haya introducido manualmente la asignación ALUMNO-OFERTA. Así pues, el primer emparejamiento que es el de ALUMNO-OFERTA, puede realizarse bien por medio del algoritmo diseñado para ello, o bien por medio de la asignación manual de alumnos y ofertas (útil en caso de que haya alumnos que al finalizar la ejecución se hayan quedado sin asignar, alumnos que independientemente de su nota media necesitan ser asignados antes que los demás por algún tipo de problema, etc.). El fichero CSV que leerá esta función tendrá el siguiente formato:

DNI;CodigoOferta

(Ejemplo: 12345678;15).

- DNI: Representa el DNI del alumno a emparejar
- CodigoOferta: Representa el código de la oferta a asignarle. Recordemos que dentro de la base de datos las ofertas se identifican con un código único. A la hora de generar este archivo CSV para realizar una inserción manual, el usuario podrá comprobar los códigos de Ofertas que tiene cada una en el archivo Ofertas Sin Asignar (ofertas que no tienen alumno asignado) y que describimos anteriormente.

En segundo lugar tenemos **leerProfesorManual**, que recibirá como parámetro la dirección de un archivo en formato CSV en el que el usuario haya introducido manualmente la asignación PROFESOR-ASIGNACIÓN. De esta forma, el segundo emparejamiento que corresponde con asignar profesores a ofertas que ya tienen asignadas un alumno, se puede realizar de dos maneras, bien por medio del algoritmo diseñado para ello (con el algoritmo suma o con el algoritmo prioridad), o bien por medio de la asignación manual de profesores y ofertas (útil en caso de que haya profesores que no hayan podido tutorizar todas las ofertas que indicaron, o profesores que por alguna razón necesitan ser asignados antes que los demás, etc.). El fichero CSV que leerá esta función tendrá el siguiente formato:

DNI;ListaCodigosOfertas

(Ejemplo: 12345678;15,18,34,156)

- DNI: Representa el DNI del profesor.

- ListaCodigosOfertas: Representa la lista de ofertas que ese profesor va a tutorizar. Esta lista de ofertas deben de estar separadas por "," como se indica en el ejemplo, y deben de ser códigos de ofertas que ya tienen un alumno asignado pero se encuentran sin tutorizar. El usuario podrá consultar la lista de códigos de ofertas que puede asignar manualmente a un profesor viendo el archivo de OfertasSinTutorizar.xlsx.

Es importante destacar que el fichero CSV que se genere para proceder a la asignación manual, tanto a la hora de insertar alumnos como a la hora de insertar profesores, el DNI que se indique solo debe de contener la parte numérica, excluyendo la letra puesto que como dijimos antes, internamente la gestión de los DNI tanto de los alumnos como de los profesores solo tiene en cuenta los caracteres numéricos. En caso de que se inserte alguna letra alfabética en el DNI de alguno de estos dos archivos, el programa la eliminará. Así pues, en caso de introducir un DNI con alguna letra, será la aplicación quien gestione ese error y la elimine para proseguir de forma normal, pero es importante que el lector sea consciente del procedimiento.

Por último, mencionar un último método que se encuadra dentro de esta clase, y aunque podría haberse creado dentro de cualquier otra, se metió dentro de esta debido a que aunque no se encarga de lectura de ficheros si tiene algo en común con el resto de métodos comentados. Este denominador común es que también interacciona con Excel, solo que en lugar de en funciones de lectura, lo hace en funciones de escritura. Es utilizado para generar los ficheros Excel correspondientes a los alumnos sin asignar, a los alumnos de intercambio, etc. y es **GENERAXLSX**.

Dentro de él, en función de qué tipo de archivo queramos crear, tendremos unas líneas de código u otras. En función de si es un tipo u otro, se utilizará unas clases u otras (Aquí es donde entran en juego las clases Asignación, Asignación Final y oferta dentro de Lógica de Negocio) y se exportará en el Excel unas

columnas en base al tipo de archivo a generar. Así pues, podríamos dividir esta función en las siguientes partes, en las que se ejecutará una u otra en función de un parámetro llamado “tipo” que recibirá la función y en el que se especificará que parte de código ejecutar.

- ALUMNOS DE INTERCAMBIO

  Recibe como parámetro una lista de alumnos y los exporta al Excel con las siguientes columnas: Año matrícula, Asignatura matrícula, Asignatura, Nombre, Plan, Nombre Plan, DNI, Alumno y Media.

- OFERTAS SIN ASIGNAR

  Recibe como parámetro una lista de ofertas y las exporta al Excel con las siguientes columnas: Código de oferta, Colegio, Localidad, Provincia y Plaza.

- OFERTAS ASIGNADAS

  Recibe como parámetro una lista de asignaciones y exporta a Excel las siguientes columnas: DNI, Nombre, CodigoPracticum, Colegio, Localidad y Provincia.

- ALUMNOS SIN ASIGNAR

  Recibe como parámetro una lista de alumnos que todavía no han sido asignados a ninguna plaza. En función del Practicum matriculado donde esté la lista de alumnos recibida generará un Excel llamado de 3 posibles formas:

  - PI_AlumnosSinAsignar,
  - PII_AlumnosSinAsignar.
  - PII_lumnosSinAsignar.

  Cada uno del anterior Excel tendrá las siguientes columnas: DNI, Código de Practicum, CodigoGrado, Nombre, Nota.

- OFERTAS SIN TUTORIZAR

  Recibe como parámetro la lista de ofertas que han sido ya

asignadas a un alumno, pero que todavía no están tutorizadas por ningún profesor y exporta a Excel las siguientes columnas: Código de Oferta, DNI del alumno, Código de Practicum, Colegio, Localidad y la Provincia.

- RESULTADO FINAL

  Recibe como parámetro la lista de asignaciones finales, es decir, la lista de ofertas que tienen asignadas un alumno, y además ya se encuentran tutorizadas por un profesor en concreto. Exportará un Excel con las siguientes columnas: DNI del profesor, Nombre del profesor, Código practicum del alumno asignado a esa oferta, DNI del alumno, nombre del alumno, plaza, colegio, localidad y provincia.

5.2.3. ModeloBBDD.java

Se trata de la clase más corta del paquete, por lo que su funcionamiento es muy sencillo. Aquí solo se especifica la dirección de la base de datos en la que se encuentran las tablas previamente cargadas, para que la aplicación pueda interactuar a la hora de mandar información y recibir información, es decir, insertar y eliminar tuplas, así como modificarlas y obtener información acerca de ellas. No solo se especifica la dirección de la base de datos, sino que también se indica el usuario con el cual se quiere acceder y la contraseña que tiene.

```
public class ModeloBBDD extends ModeloBBDDAbstracto {

        public ModeloBBDD() {
                super("127.0.0.1/coles7", "root", "");
        }
```

Fig. 36. Código de la clase ModeloBBDD().

Como se ve en la imagen, en nuestro caso para realizar las pruebas, la dirección utilizada es 127.0.0.1, o lo que es lo mismo, “localhost”. “coles7” es el nombre de la base de datos dentro de phpMyAdmin, mientras que “root” es el usuario, y como puede verse, no tiene contraseña.

5.2.4. ModeloBBDDAbstracto.java

Dentro de lo que hemos denominado BackEnd, es decir, todas las clases que componen la lógica de negocio, esta es con más de 2800 líneas la clase que más líneas de código tiene con diferencia. Como puede verse en la imagen anterior, esta es la clase padre de ModeloBBDD, y es esta la que realiza la conexión, solo que la clase ModeloBBDD fue separada del resto de cara a que si alguna vez el usuario quiere cambiar de dirección de base de datos, no tenga que buscar entre más de mil líneas de código, sino que pueda cambiarla de forma fácil y sencilla. El constructor de esta clase es precisamente el que llama la imagen anterior, tal que así:

```
public class ModeloBBDDAbstracto {
     private Connection conexion;
     private String direccionBBDD;
     private String usuario;
     private String contraseña;


     public ModeloBBDDAbstracto(String direccionBBDD, String usuario, String contraseña) {
                this.conexion = null;
                this.direccionBBDD = direccionBBDD;
                this.usuario = usuario;
                this.contraseña = contraseña;
        }

    public Connection getConexion() {
        return conexion;
    }

    public void setConexion(Connection conexion) {
        this.conexion = conexion;
    }

    public String getDireccionBBDD() {
```

Fig. 36. Constructor y métodos principales de la clase ModeloBBDDAbstracto().

El método más importante sin duda dentro de esta clase es el que se encarga de establecer la conexión. Esta función, que hace uso de la librería MySQL JDBC Driver, y que tuvo que ser instalada antes de programar la aplicación, se encarga de establecer la conexión con la base de datos especificada en función de los valores que se pusieron en ModeloBBDD. Su estructura es esta:

```
public void establecerConexion() {
    try {

        Class.forName("com.mysql.jdbc.Driver").newInstance();
            System.out.println("Registro exitoso");

    } catch (Exception e) { System.out.println(e.toString()); }

            conexion= null;

            try {
                    conexion = DriverManager.getConnection(
                                    "jdbc:mysql://" + direccionBBDD + "?" + "user="
                                            + usuario + "&password=" + contraseña);
                    System.out.println(conexion.toString());
                    conexion.setAutoCommit(false);

            } catch (SQLException ex) {
                System.out.println("SQLException: " + ex.getMessage());
            }
    }
```

Fig. 36. Código del método establecerConexion().

La importancia de este método radica en que cada vez que hagamos uso de alguna de las funciones que mostraremos más adelante que impliquen algún tipo de conexión con la base de datos ya sea de consultar, insertar, modificar o eliminar, hay que llamar a esta función primero, para que pueda conectarse a la BBDD correctamente.

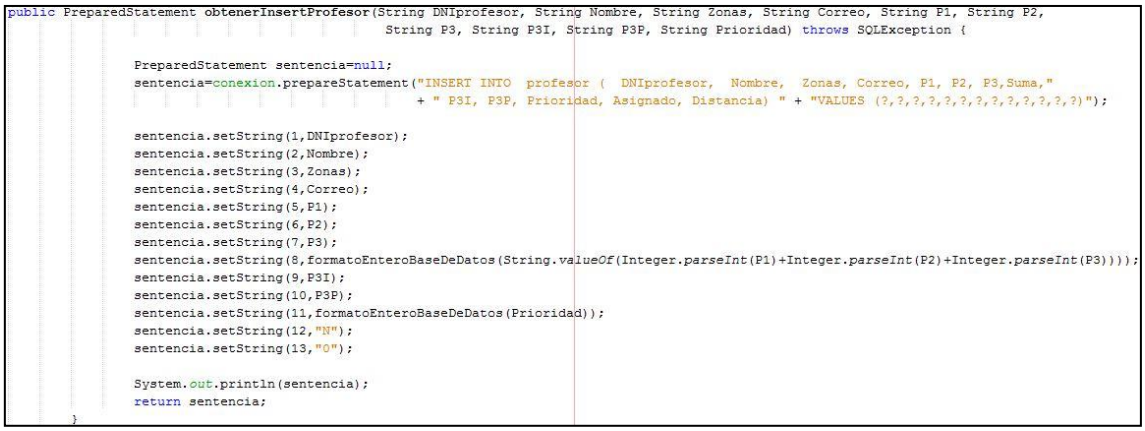
A continuación se presentan una serie de métodos cuya única función es ser llamados a la hora de realizar una inserción en la base de datos, puesto que reciben los atributos que se quieren subir a una determinada tabla, y devuelve la cadena de caracteres concreta en formato SQL que hay que ejecutar para que se produzca dicha inserción.

```
public PreparedStatement obtenerInsertCentro(String CodigoCentro, String Nombre, String Localidad,
                                             String Zona, String CodigoOficial, String Provincia) throws SQLException{

        PreparedStatement sentencia=null;
        sentencia=conexion.prepareStatement("INSERT INTO  centro (  CodigoCentro,  Nombre,  Localidad,  Zona,"
                                          + " CodigoOficial, Provincia) " + "VALUES (?,?,?,?,?,?)");

        sentencia.setString(1,CodigoCentro);
        sentencia.setString(2,Nombre);
        sentencia.setString(3,Localidad);
        sentencia.setString(4,Zona);
        sentencia.setString(5,CodigoOficial);
        sentencia.setString(6, Provincia);

        System.out.println(sentencia);
        return sentencia;
    }
```

Fig. 37. Código del método obtenerInsertCentro().

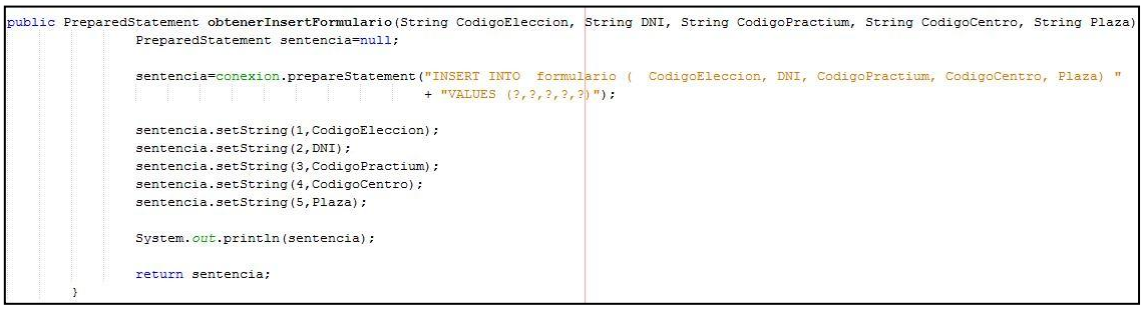
```
public PreparedStatement obtenerInsertGrado(String CodigoGrado, String Nombre) throws SQLException{
        PreparedStatement sentencia=null;

        sentencia=conexion.prepareStatement("INSERT INTO  grado (  CodigoGrado,  Nombre) "
                                          + "VALUES (?,?)");

        sentencia.setString(1,CodigoGrado);
        sentencia.setString(2,Nombre);

        System.out.println(sentencia);

        return sentencia;
    }
```

Fig. 38. Código del método obtenerInsertGrado().



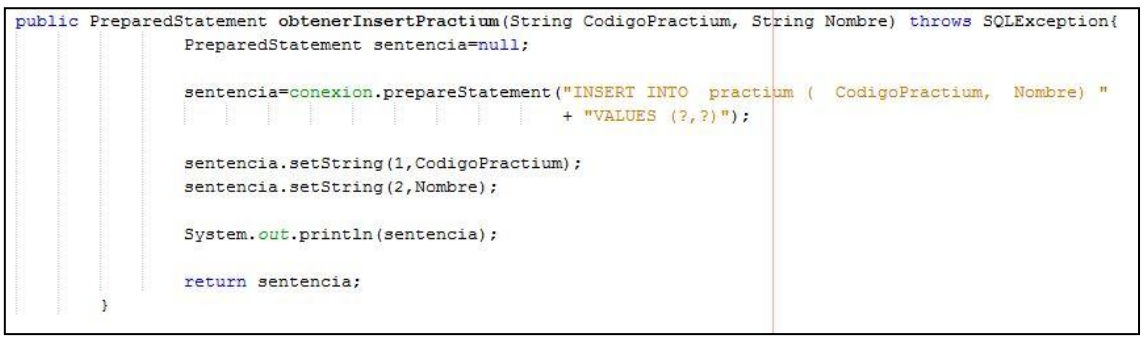
```
public PreparedStatement obtenerInsertProfesor(String DNIprofesor, String Nombre, String Zonas, String Correo, String P1, String P2,
                                               String P3, String P3I, String P3P, String Prioridad) throws SQLException {

        PreparedStatement sentencia=null;
        sentencia=conexion.prepareStatement("INSERT INTO  profesor (  DNIprofesor,  Nombre,  Zonas, Correo, P1, P2, P3,Suma,"
                                            + " P3I, P3P, Prioridad, Asignado, Distancia) " + "VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?)");

        sentencia.setString(1,DNIprofesor);
        sentencia.setString(2,Nombre);
        sentencia.setString(3,Zonas);
        sentencia.setString(4,Correo);
        sentencia.setString(5,P1);
        sentencia.setString(6,P2);
        sentencia.setString(7,P3);
        sentencia.setString(8,formatoEnteroBaseDeDatos(String.valueOf(Integer.parseInt(P1)+Integer.parseInt(P2)+Integer.parseInt(P3))));
        sentencia.setString(9,P3I);
        sentencia.setString(10,P3P);
        sentencia.setString(11,formatoEnteroBaseDeDatos(Prioridad));
        sentencia.setString(12,"N");
        sentencia.setString(13,"0");

        System.out.println(sentencia);
        return sentencia;
    }
```

Fig. 39. Código del método obtenerInsertProfesor().



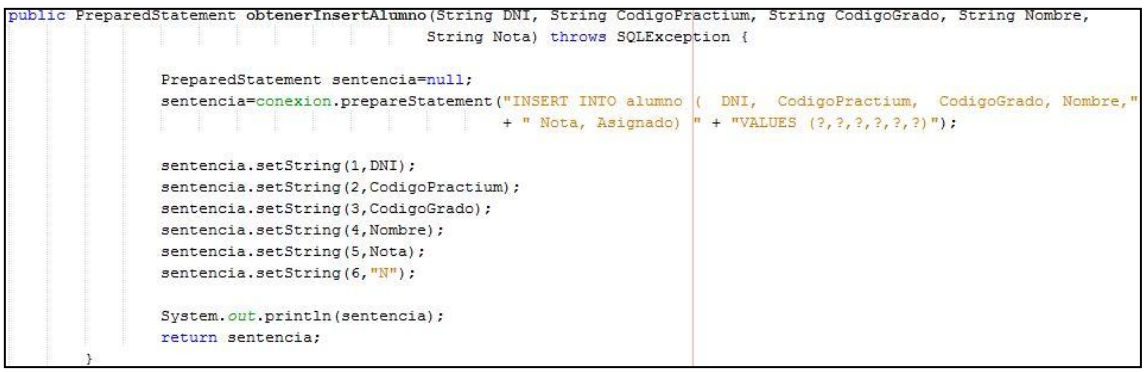
```
public PreparedStatement obtenerInsertFormulario(String CodigoEleccion, String DNI, String CodigoPractium, String CodigoCentro, String Plaza)
        PreparedStatement sentencia=null;

        sentencia=conexion.prepareStatement("INSERT INTO  formulario (  CodigoEleccion, DNI, CodigoPractium, CodigoCentro, Plaza) "
                                            + "VALUES (?,?,?,?,?)");

        sentencia.setString(1,CodigoEleccion);
        sentencia.setString(2,DNI);
        sentencia.setString(3,CodigoPractium);
        sentencia.setString(4,CodigoCentro);
        sentencia.setString(5,Plaza);

        System.out.println(sentencia);

        return sentencia;
    }
```

Fig. 40. Código del método obtenerInsertFormulario().



```
public PreparedStatement obtenerInsertPractium(String CodigoPractium, String Nombre) throws SQLException{
        PreparedStatement sentencia=null;

        sentencia=conexion.prepareStatement("INSERT INTO  practium (  CodigoPractium,  Nombre) "
                                            + "VALUES (?,?)");

        sentencia.setString(1,CodigoPractium);
        sentencia.setString(2,Nombre);

        System.out.println(sentencia);

        return sentencia;
    }
```

Fig. 41. Código del método obtenerInsertPractium().



```
public PreparedStatement obtenerInsertAlumno(String DNI, String CodigoPractium, String CodigoGrado, String Nombre,
                                             String Nota) throws SQLException {

        PreparedStatement sentencia=null;
        sentencia=conexion.prepareStatement("INSERT INTO alumno (  DNI,  CodigoPractium,  CodigoGrado, Nombre,"
                                            + " Nota, Asignado) " + "VALUES (?,?,?,?,?,?)");

        sentencia.setString(1,DNI);
        sentencia.setString(2,CodigoPractium);
        sentencia.setString(3,CodigoGrado);
        sentencia.setString(4,Nombre);
        sentencia.setString(5,Nota);
        sentencia.setString(6,"N");

        System.out.println(sentencia);
        return sentencia;
    }
```

Fig. 42. Código del método obtenerInsertAlumno().

```
public PreparedStatement obtenerInsertAsignacion(String DNI, String CodigoPractium, String CodigoCentro,
                                                 String CodigoOferta) throws SQLException, IOException {

        PreparedStatement sentencia=null;
        sentencia=conexion.prepareStatement("INSERT INTO asignacion (  DNI,  CodigoPractium,  CodigoCentro, "
                                            + "CodigoOferta, Tutorizada, SumaOfertasCentro) " + "VALUES (?,?,?,?,?,?)");

        sentencia.setString(1,DNI);
        sentencia.setString(2,CodigoPractium);
        sentencia.setString(3,CodigoCentro);
        sentencia.setString(4,CodigoOferta);
        sentencia.setString(5,"N");
        sentencia.setString(6,"");

        System.out.println(sentencia);
        return sentencia;
    }
```

Fig. 43. Código del método obtenerInsertAsignacion().

```
public PreparedStatement obtenerInsertAsignacionProfesor(String DNI, String CodigoPractium, String CodigoCentro,
                                                         String CodigoOferta, String DNIprofesor) throws SQLException {

        PreparedStatement sentencia=null;
        sentencia=conexion.prepareStatement("INSERT INTO asignacionprofesor ( DNI,  CodigoPractium,  CodigoCentro, "
                                            + "CodigoOferta, DNIprofesor) " + "VALUES (?,?,?,?,?)");

        sentencia.setString(1,DNI);
        sentencia.setString(2,CodigoPractium);
        sentencia.setString(3,CodigoCentro);
        sentencia.setString(4,CodigoOferta);
        sentencia.setString(5,DNIprofesor);

        System.out.println(sentencia);
        return sentencia;
    }
```

Fig. 44. Código del método obtenerInsertAsignacionProfesor().

```
public PreparedStatement obtenerInsertOferta(String CodigoOferta, String CodigoCentro,
                                             String Nombre) throws SQLException {

    PreparedStatement sentencia=null;
    sentencia=conexion.prepareStatement("INSERT INTO oferta (  CodigoOferta,  CodigoCentro,  Nombre, Asignada) "
                                        + "VALUES (?,?,?,?)");

    sentencia.setString(1,CodigoOferta);
    sentencia.setString(2,CodigoCentro);
    sentencia.setString(3,Nombre);
    sentencia.setString(4,"N");

   System.out.println(sentencia);
   return sentencia;
}
```

Fig. 45. Código del método obtenerInsertOferta().

Como puede verse, anteriormente se han mostrado 9 trozos de código, uno por cada entidad que existe en la base de datos y que puede consultarse en el diagrama explicado anteriormente.

Sin embargo, hay una serie de métodos que también reciben una serie de atributos, y que también devuelven la consulta SQL exacta que hay que realizar, pero esta no se refiere a realizar una inserción, sino a realizar una modificación, y que vale la pena mencionar debido a su importancia. En primer lugar, como indicamos en el diseño de la base de datos, a la hora de realizar la asignación ALUMNO-OFERTA, se

hace uso de la tabla formulario, consultando si las elecciones de los alumnos están disponibles. Pues bien, cuando una elección de algún alumno está disponible, esa plaza concreta es asignada, realizándose dos cosas. La primera de ellas es crear una nueva entrada en la tabla ASIGNACIÓN, indicando el DNI, CodigoCentro, Plaza, etc. La segunda de esas cosas es MODIFICAR la tabla OFERTA, cambiando el flag "Asignada" a valor 'S' en lugar de a 'N', de forma que esa oferta que acaba de ser asignada, no la tenga en cuenta el programa para el resto de asignaciones. Esto se consigue por medio de la siguiente consulta SQL cada vez que un formulario es asignado:

```
public PreparedStatement cambiarEstadoOferta(String CodigoOferta) throws SQLException{
        PreparedStatement sentencia=null;

        sentencia=conexion.prepareStatement("UPDATE oferta SET Asignada='S' WHERE CodigoOferta = '" + CodigoOferta + "'");

        System.out.println(sentencia);


        return sentencia;

    }
```

Fig. 46. Código del método cambiarEstadoOferta().

Así mismo, cuando un alumno es asignado a una oferta, no solo se cambia el flag de Oferta ya comentado, sino también otro flag similar en "Alumno", de modo que ese alumno también sea descartado para futuras asignaciones, tal que así:

```
public PreparedStatement cambiarEstadoAlumno(String DNI) throws SQLException{
        PreparedStatement sentencia=null;

        sentencia=conexion.prepareStatement("UPDATE alumno SET Asignado='S' WHERE DNI = '" + DNI + "'");

        System.out.println(sentencia);
        return sentencia;

    }
```

Fig. 47. Código del método cambiarEstadoAlumno().

Como puede verse, los dos últimos métodos que acabamos de mencionar son utilizados en la asignación ALUMNO-OFERTA. Pues bien, hay otros dos métodos también dedicados a modificar las tablas, y que serán utilizados en el siguiente paso de la aplicación que es la asignación entre PROFESOR y ASIGNACIÓN. El primero de estos métodos, de forma similar al mencionado del alumno, se encarga de cambiar el flag de profesor estableciéndolo como Asignado='S', de forma que ese profesor no sea tenido en cuenta para el resto de la ejecución:

```
public PreparedStatement cambiarEstadoProfesor(String DNIprofesor) throws SQLException{
        PreparedStatement sentencia=null;

        sentencia=conexion.prepareStatement("UPDATE profesor SET Asignado='S' WHERE "
                                            + "DNIprofesor = '" + DNIprofesor + "'");

        System.out.println(sentencia);
        return sentencia;
    }
```

Fig. 48. Código del método cambiarEstadoProfesor().

Así mismo, de forma análoga al caso anterior, cada vez que una asignación (Alumno y Oferta ya asignados) es tutorizada por un determinado profesor, esa asignación ya no puede aparecer como "posible" para el resto de los profesores, puesto que ya se encuentra tutorizada. Es por ello por lo que cada vez que realicemos internamente una nueva "tutorización", esa oferta debe ser marcada con el flag Tutorizada='S', tal que así:

```
public PreparedStatement cambiarEstadoAsignacion(String CodigoOferta) throws SQLException{

        PreparedStatement sentencia=null;
        sentencia=conexion.prepareStatement("UPDATE asignacion SET Tutorizada='S'"
                + " WHERE CodigoOferta = '" + CodigoOferta + "'");

        System.out.println(sentencia);
        return sentencia;
    }
```

Fig. 49. Código del método cambiarEstadoAsignación().

En esta entidad también existirán una serie de métodos de comprobación, para evitar errores en las inserciones o modificaciones. El primero de esos métodos se encarga de comprobar si un determinado alumno existe dentro de la base de datos, es decir, dentro de la tabla "Alumno". Recibe como parámetro el DNI y la función devuelve true si ese DNI existe en la base de datos, o false en caso contrario.

```
public boolean existeAlumno(String DNI) {
    establecerConexion();

        try {
            Statement cmd = conexion.createStatement();
            ResultSet rs1 = cmd.executeQuery("SELECT * FROM alumno WHERE DNI= '" + DNI + "'");

            int count = 0;
            while (rs1.next())
                count++;

            return count==1;
             }
```

Fig. 50. Código del método existeAlumno().

También existe un método para comprobar si un determinado centro existe. En este caso no recibe el DNI, sino el CÓDIGO OFICIAL del centro, y devuelve true si existe dentro de la base de datos, o false en caso contrario.

```
public boolean existeCentro(String CodigoOficial) {
    establecerConexion();

        try {
            Statement cmd = conexion.createStatement();

            ResultSet rs1 = cmd.executeQuery("SELECT * FROM centro WHERE CodigoOficial= '" + CodigoOficial + "'");

            int count = 0;
            while (rs1.next())
                count++;

            return count==1;

        }   catch (SQLException e) {
```

Fig. 51. Código del método existeCentro().

Como puede verse en las dos imágenes anteriores, la primera línea de código antes de interactuar con la base de datos, debe de ser siempre **establecerConexion()**. Del mismo modo, cabe mencionar que tanto existeAlumno(), como existeCentro(), son utilizados por medio del método **leerFormularios() y leerMatriculas()** dentro de LecturaFicheros.java que ya comentamos anteriormente, precisamente para saber si el formulario que se acaba de leer del Excel, corresponde a un alumno real, y dicho alumno ha solicitado un centro real, como se ve en la siguiente imagen, ya que en caso contrario, esa línea del fichero correspondiente a ese alumno o ese centro no sería tenida en cuenta, y saltaría a la siguiente línea en la lectura del fichero, sin realizar ningún tipo de acción en la anterior (como si fuese una línea en blanco). El código referente a este proceso de selección sería tal que así:

```
if (mod.existeAlumno(DNI) && mod.existeCentro(CodigoOficial) && !Plaza.equals("")) {

    mod.crearFormulario(CodigoEleccion, DNI, CodigoOficial,Plaza);
    Plaza="";
}
```

Fig. 52. Código que selecciona que líneas del fichero debe tener en cuenta.

Otro de los métodos auxiliares que tenemos es obtenerFilas(). Este método recibe el tipo de tabla de la que se quiere consultar el número exacto de filas que contiene (p. ej. Centro, alumno, etc.), y devuelve un valor entero con dichas filas. Este método es usado por ejemplo a la hora de insertar centros, ya que aunque a la hora de insertar la primera tanda de centros (p. ej. Guadalajara), el CODIGO CENTRO no presenta ninguna dificultad a la hora de crearlo, puesto que empezará en 1 y terminará cuando termine de leer las filas, después de la primera tanda, al insertar los centros de la comunidad de Madrid, el CODIGO CENTRO para cada uno de los centros ya no empieza en 1, sino que se consultará a este método el número de filas que tiene, y se seguirá dicha numeración de forma correlativa. Por ejemplo si Guadalajara tiene 75 centros, este método se utiliza para saber que los códigos centros de la comunidad de Madrid empezarán en 76. El formato del método es el siguiente:

```
public int obtenerFilas(String tipo) {
        establecerConexion();

        try {
            Statement cmd = conexion.createStatement();

            ResultSet rs1 = cmd.executeQuery("SELECT * FROM " + tipo);

            int count = 1;
            while (rs1.next())
                count++;

            return count;

             }
```

Fig. 53. Código del método obtenerFilas().

A continuación explicaremos el trío de métodos que se encargan de la creación de un Alumno, de un Centro, y de un Practicum. Los tres han sido agrupados en la

misma explicación, porque los 3 son llamados desde el método ya comentado leerMatriculas () en LecturaFicheros.java, ya que por cada línea del archivo de las matrículas de los alumnos que leemos, se obtienen estas 3 informaciones, que son el grado, el Practicum, y el alumno que pertenece a ese Practicum y a ese Grado.

El método encargado de crear un Practicum únicamente recibirá dos valores, que son el CodigoPractium y el Nombre de ese Practicum, y antes de insertarlo en la BBDD, comprobará que no ha sido ya insertado, tal que así:

```
public boolean crearPractium(String CodigoPractium, String Nombre) {
    establecerConexion();

    try {
        Statement cmd = conexion.createStatement();  //Para nombre

        //POR NOMBRE
        ResultSet rs1 = cmd.executeQuery("SELECT * FROM practium WHERE CodigoPractium= '" + CodigoPractium + "'");

        rs1.next();

            if (rs1.getRow()>0) {
                return false;
            }

        PreparedStatement insertPractium=obtenerInsertPractium(CodigoPractium, Nombre);
        insertPractium.executeUpdate();     //crea centro

        conexion.commit(); //Se indica que se deben aplicar los cambios en la base de datos
        return true;

    }
```

Fig. 54. Código del método crearPractium().

El método encargado de crear un Grado recibirá también dos valores, que son el Código del grado (G200, G210, etc.) y el nombre de ese grado, y al igual que sucede con el anterior, antes de insertarlo, comprobaremos que no existe ya en la BBDD, para no tener filas repetidas, de esta forma:

```
public boolean crearGrado(String CodigoGrado, String Nombre) {

    establecerConexion();

    try {
        Statement cmd = conexion.createStatement();  //Para nombre

        //POR NOMBRE
        ResultSet rs1 = cmd.executeQuery("SELECT * FROM grado WHERE CodigoGrado= '" + CodigoGrado + "'");

        rs1.next();

            if (rs1.getRow()>0) {
                return false;
            }
        PreparedStatement insertGrado=obtenerInsertGrado(CodigoGrado, Nombre);
        insertGrado.executeUpdate();

        conexion.commit(); //Se indica que se deben aplicar los cambios en la base de datos
        return true;

    }
```

Fig. 55. Código del método crearGrado().

Por último, el método encargado de crear un alumno recibir 5 atributos que serán el DNI del alumno, su nombre, su nota media, el código del grado al que pertenece (mismo código Grado que el de crearGrado()), y el código del Practicum (mismo código del Practicum que el de crearPracticum()). Al igual que sucede con los dos anteriores, antes de insertarlo se comprobará que no existe en la base de datos un alumno con el mismo DNI y mismo código de Practicum, tal que así:

```
public boolean crearAlumno(String DNI, String CodigoPractium, String CodigoGrado, String Nombre, String Nota) {
    establecerConexion();

    try {
        Statement cmd = conexion.createStatement();  //Para nombre

        ResultSet rs1 = cmd.executeQuery("SELECT * FROM alumno WHERE DNI= '" + DNI + "' AND CodigoPractium= '"
                                         + CodigoPractium + "'");
        rs1.next();

        if (rs1.getRow()>0)
            return false;

        PreparedStatement insertAlumno=obtenerInsertAlumno(DNI, CodigoPractium, CodigoGrado, Nombre, Nota);
        insertAlumno.executeUpdate();

        conexion.commit(); //Se indica que se deben aplicar los cambios en la base de datos
        return true;
    }
```

Fig. 56. Código del método crearGrado().

Ahora entraremos a comentar otro de los métodos más importantes de esta clase, que es **CREAR FORMULARIO**. Crear Formulario, como dijimos antes, es llamado desde leerFormulario() cada vez que se quiere insertar un formulario concreto, es decir, una vez por cada línea de datos que tenga el Excel de la elección de los alumnos. Este método simplemente inserta en la base de datos, dentro de la tabla formulario, 4 valores que son el código de la elección (por orden de preferencia, el DNI, el código centro, y el nombre de la plaza). Pero si recordamos el diseño da la base de datos, en el archivo de las elecciones de los alumnos se nos indica el código oficial de cada centro pero no el código centro de 1 a N centros que es con lo que nosotros trabajamos a la hora de referenciar, por lo que antes de crear el formulario hay que hacer dos cosas:

- Obtener el código Practicum del alumno que ha rellenado dicho formulario.
- Obtener el código del centro en base al código oficial que se nos ha proporcionado.

Una vez hechas esas dos tareas, ya podríamos proceder a la inserción del formulario en la BBDD. A continuación se muestra un pequeño diagrama que explica el proceso de creación del formulario:

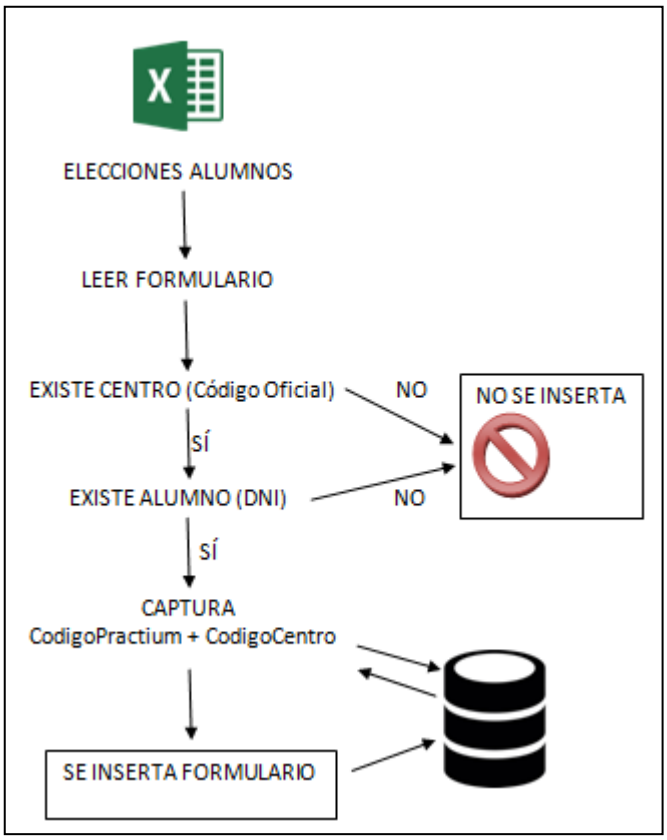


Fig. 57. Diagrama representativo de la creación de Formularios en la BBDD.

Así pues, el código que representa la creación del formulario expuesta en el anterior diagrama es el siguiente:

```
public boolean crearFormulario(String CodigoEleccion, String DNI, String CodigoOficial, String Plaza) {
 establecerConexion();
 String CodigoCentro,CodigoPractium;

try {
    Statement cmd = conexion.createStatement();

    //Para obtener el codigo de Practium del alumno que estamos buscando
    ResultSet rs1 = cmd.executeQuery("SELECT CodigoPractium FROM alumno WHERE DNI= '" + DNI + "'");
    rs1.next();
    CodigoPractium=rs1.getString("CodigoPractium");

    ResultSet rs2 = cmd.executeQuery("SELECT CodigoCentro FROM centro WHERE CodigoOficial= '" + CodigoOficial + "'");
    rs2.next();
    CodigoCentro=rs2.getString("CodigoCentro");


    PreparedStatement insertFormulario=obtenerInsertFormulario(CodigoEleccion, DNI, CodigoPractium, CodigoCentro, Plaza);
    insertFormulario.executeUpdate();

    conexion.commit(); //Se indica que se deben aplicar los cambios en la base de datos
    return true;
```

Fig. 58. Código del método crearFormulario().

Otro de los grandes métodos a analizar es **CREAR CENTRO GUADALAJARA**. Como dijimos anteriormente, en los ficheros de los centros habrá dos archivos, y los dos tendrán distinto formato, lo que implica que la llamada a la hora de crear un centro, hagamos una bifurcación en función del tipo del tipo de centro a crear (Guadalajara o Comunidad de Madrid). En cuanto a la creación de GUADALAJARA, recibe el código del Centro (que ya se ha encargado leerCentrosGuadalajara() de establecerlo), el nombre del centro, la localidad y la zona en la que se encuentra, el código oficial, y otras 11 variables de formato entero, una por cada tipo de oferta que puede haber publicado ese centro, en la que se indicarán si han creado 3 ofertas de un tipo, 8 de otro, etc. Acto seguido, lo que se hace es comprobar si ese nombre de centro y esa localidad ya existen en la base de datos, puesto que si existen no se insertará. En caso de que dicho centro exista, se harán dos cosas. En primer lugar se ejecutará la consulta que implica la creación del centro, y en segundo lugar se procederá (sin salir del método) a la creación de tuplas en la tabla Oferta (una por cada oferta que ese centro haya publicado) con el formato estandarizado que antes hablábamos.

Se muestran a continuación partes de código del método crearCentroGuadalajara(). En primer lugar como puede verse, la creación del centro:

```
public boolean crearCentroGuadalajara(String Codigo, String Nombre, String Localidad, String Zona,String EP, String EI, String EF, S
    establecerConexion();
    String loc;
    int CodigoOferta;

    try {
        Statement cmd = conexion.createStatement();  //Para nombre

        //POR NOMBRE
        ResultSet rs1 = cmd.executeQuery("SELECT * FROM centro WHERE Nombre= '" + Nombre + "'");

        while (rs1.next()) {

            loc=rs1.getString(3);

            if (loc.equals(Localidad)) {
                return false;
            }
        }

        PreparedStatement insertCentro=obtenerInsertCentro(Codigo, Nombre, Localidad, Zona, CodigoOficial);
        insertCentro.executeUpdate();     //crea centro
```

Fig. 59. Parte del código del método crearCentroGuadalajara() dedicado a la creación de centros.

Una vez creado el centro, se procedería a la creación de cada una de las ofertas:

```
if (!EP.equals("")) {
    EPi=Integer.parseInt(EP);
    while (EPi>0) {
        conexion.close();
       CodigoOferta=obtenerFilas("oferta");
       establecerConexion();
       PreparedStatement insertOferta=obtenerInsertOferta(String.valueOf(CodigoOferta), Codigo, "Educacion primaria");
       insertOferta.executeUpdate();     //crea oferta
       EPi--;
    }
}

if (!EI.equals("")) {
    EIi=Integer.parseInt(EI);
    while (EIi>0) {
        conexion.close();
       CodigoOferta=obtenerFilas("oferta");
       establecerConexion();
       PreparedStatement insertOferta=obtenerInsertOferta(String.valueOf(CodigoOferta), Codigo, "Educacion infantil");
       insertOferta.executeUpdate();     //crea oferta
       EIi--;
```

Fig. 60. Parte del código del método crearCentroGuadalajara() dedicado a la creación de ofertas.

Así pues, una vez que haya finalizado la ejecución del método, no solo se habrá insertado el centro, sino también todas las ofertas que este ha publicado.

En segundo lugar tenemos el método de **CREAR CENTRO MADRID**. En este, a diferencia del anterior, no se le llama una única vez por centro. Puesto que el formato del fichero de la comunidad de Madrid cada línea corresponde con una oferta (repitiéndose posiblemente el centro con la línea anterior o la siguiente), a esta función se la llamará una vez por cada oferta y no por cada centro, todo ello con el control de que solo insertará el centro en la base de datos cuando este no haya sido insertado previamente. Así pues, a este método se le indicará el código del centro, su código oficial, el nombre, la localidad y la zona donde se encuentra, y la única oferta que publica (de ahí que haya que llamar varias veces a esta función). Primero se comprobará que no existe ese nombre de centro en esa misma localidad:

```
public boolean crearCentroMadrid(String Codigo, String Nombre, String Localidad, String Zona, String oferta, String CodigoOficial) {
    establecerConexion();
    String loc;
    String nuevaOf="";
    int CodigoOferta;
    boolean respuesta=true, ejecutarOferta=true;

    try {
        Statement cmd = conexion.createStatement();  //Para nombre

        //POR NOMBRE
        ResultSet rs1 = cmd.executeQuery("SELECT * FROM centro WHERE Nombre= '" + Nombre + "'");

        while (rs1.next()) {
            loc=rs1.getString(3);
            if (loc.equals(Localidad)) {

                respuesta=false;
                Codigo=String.valueOf(Integer.parseInt(Codigo)-1);
            }
        }
```

Fig. 61. Parte del código del método crearCentroMadrid() comprobadora de la existencia del centro.

Una vez comprobada la existencia del centro, se hará la inserción de OFERTAS. En primer lugar se le dará al nombre de la plaza que se oferta un nombre estandarizado:

```
if (oferta.equals("Maestro: Educacion Primaria") || oferta.equals("Educacion Primaria"))
    nuevaOf="Educacion primaria";
else if (oferta.equals("Maestro: Educacion Infantil (de 3 a 6 anos)") || oferta.equals("Educacion Infantil (de 3 a 6 anos)"))
    nuevaOf="Educacion infantil";
else if (oferta.equals("Maestro: Educacion Fisica") || oferta.equals("Educacion Fisica"))
    nuevaOf="Educacion fisica";
else if (oferta.equals("Maestro: Educacion Musical") || oferta.equals("Educacion Musical"))
    nuevaOf="Educacion musical";
else if (oferta.equals("Maestro: Lengua Extranjera") || oferta.equals("Lengua Extranjera"))
    nuevaOf="Lengua extranjera";
else if (oferta.equals("Maestro: Pedagogia Terapeutica") || oferta.equals("Pedagogia Terapeutica"))
    nuevaOf="Pedagogia terapeutica";
else if (oferta.equals("Maestro: Audicion y Lenguaje") || oferta.equals("Audicion y Lenguaje"))
    nuevaOf="Audicion y lenguaje";

else if (oferta.equals("Maestro: Educacion Primaria - B") || oferta.equals("Educacion Primaria - B"))
    nuevaOf="Educacion primaria - B";
else if (oferta.equals("Maestro: Educacion Infantil (de 3 a 6 anos) - B") || oferta.equals("Educacion Infantil (de 3 a 6 anos) - B"))
    nuevaOf="Educacion infantil - B";
else if (oferta.equals("Maestro: Educacion Fisica - B") || oferta.equals("Educacion Fisica - B"))
    nuevaOf="Educacion fisica - B";
else if (oferta.equals("Maestro: Educacion Musical - B") || oferta.equals("Educacion Musical - B"))
    nuevaOf="Educacion musical - B";

else if (oferta.equals("Maestro: Lengua Extranjera - B") || oferta.equals("Lengua Extranjera - B"))
    nuevaOf="Lengua extranjera - B";
else if (oferta.equals("Maestro: Pedagogia Terapeutica - B") || oferta.equals("Pedagogia Terapeutica - B"))
    nuevaOf="Pedagogia terapeutica - B";
else if (oferta.equals("Maestro: Audicion y Lenguaje - B") || oferta.equals("Audicion y Lenguaje - B"))
    nuevaOf="Audicion y lenguaje - B";
```

Fig. 62. Parte del código del método crearCentroMadrid() que estandariza el nombre de la oferta.

Después se procederá a la inserción del centro (en caso de que en la primera de las consultas no haya encontrado ningún centro con esas características) y la creación de la oferta, llamando al método obtenerFilas() comentado previamente, de forma que el código de la oferta vaya en correlación con los anteriores, tal que así:

```
//CREAMOS EL CENTRO
if (respuesta) {
    PreparedStatement insertCentro=obtenerInsertCentro(Codigo, Nombre, Localidad, Zona, CodigoOficial);
    insertCentro.executeUpdate();      //crea centro
}

conexion.close();
CodigoOferta=obtenerFilas("oferta");
establecerConexion();

if (ejecutarOferta) {
    PreparedStatement insertOferta=obtenerInsertOferta(String.valueOf(CodigoOferta), Codigo, nuevaOf);
    insertOferta.executeUpdate();      //crea oferta
}

conexion.commit(); //Se indica que se deben aplicar los cambios en la base de datos
```

Fig. 63. Parte del código del método crearCentroMadrid() que crea el centro y la oferta.

Así pues, la representación en DIAGRAMA de la creación de centros tanto de Madrid como de Guadalajara seguiría el siguiente formato:

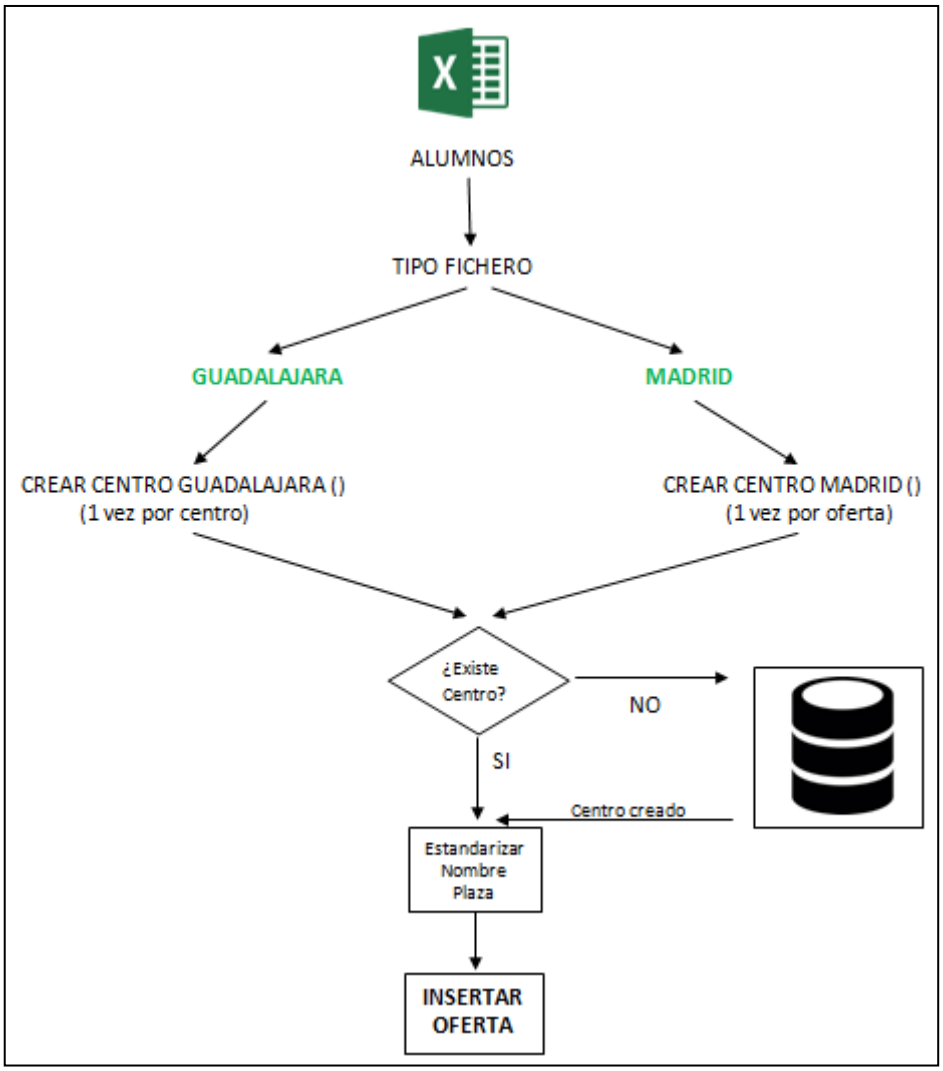


Fig. 64. Diagrama de creación de centros y ofertas en la base de datos.

El siguiente método a comentar es **ORDENAR ALUMNOS**. Este método no es sino la culminación de la primera parte del trabajo, puesto que se encarga de realizar la asignación ALUMNO-OFERTA.

Es decir, ahora ya tenemos en la base de datos almacenados todos los datos acerca de los alumnos, todos los formularios que estos han rellenado, y todas las ofertas que han publicado los centros, así que ahora toca realizar con todos esos datos la asignación ALUMNO-OFERTA, rellenando por cada una de esas asignaciones una tupla en la entidad "Asignación" como se explicó en el diseño de la base de datos.

Así pues, la asignación de alumnos será realizada por este método de la siguiente manera. Primero escogerán los alumnos matriculados en PIII (sin importar la especialidad), luego PI, y por último los alumnos de PII. Así mismo, dentro de cada tipo de Practicum, los alumnos serán ordenados de más a menos nota media. El siguiente diagrama representa dicha asignación:

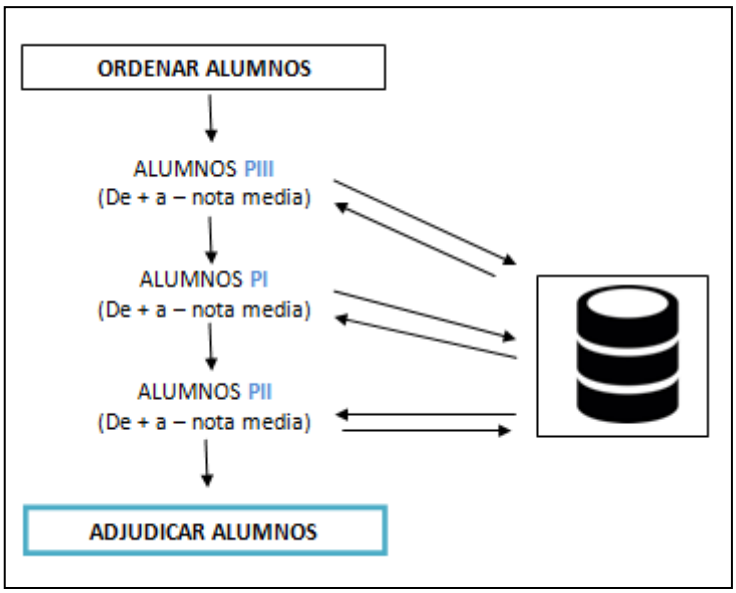


Fig. 65. Diagrama con el orden de los alumnos a asignar.

Es decir, el primero en elegir será el alumno de Practicum III con la nota media más alta, y el último el alumno matriculado en Practicum I con la nota media más baja. A continuación se muestra un ejemplo de cómo sería el código para cada uno de esos grupos de Alumnos.

Cabe mencionar que en la presente memoria se está dando por hecho que la ejecución del algoritmo se realizará de golpe con todos los alumnos de PI, de PII y de PIII. Sin embargo, la realidad es que la aplicación se correrá en diferentes momentos del curso académico, ejecutando por ejemplo primero solo los alumnos de PIII o solo los alumnos de PI. Esto no presenta problema, ya que la aplicación solo tiene en cuenta los alumnos que estén cargados en la base de datos en un momento determinado, por lo que no tiene por qué haber alumnos de los 3 tipos cargados para poder ejecutar la aplicación.

Así pues, una vez que se hayan obtenido los ficheros resultantes, entre ellos habrá un archivo en formato ".csv" con los resultados de la asignación hecha hasta ese momento, de forma que si se quiere ejecutar la totalidad de los alumnos en

diferentes momentos del curso académico se pueden cargar las asignaciones que hay hasta ese momento reflejadas en el fichero ".csv" y proseguir con el nuevo grupo de alumnos que se quiera asignar.

En la siguiente imagen se muestra como se cogerían de la base de datos los Alumnos de PIII para ser asignados, uno a uno, llamando al método ADJUDICAR ALUMNOS (que comentaremos más adelante) una vez por cada alumno. El código correspondiente a la asignación de los alumnos de PI y PII que está justo debajo sería el mismo que el expuesto pero cambiando algunos pocos detalles:

```
//CODIGOS PRACTIUM III
ResultSet rs = cmd.executeQuery("SELECT * FROM practium WHERE Nombre LIKE '%PRACTICUM III%'");
while (rs.next()) {
    codigosPractium3.add(rs.getString("CodigoPractium"));
    consultaExitosa=true;
}

if (consultaExitosa) {
    listaCodigosPractium3="CodigoPractium = '" + codigosPractium3.get(0) + "'";

    for (int i = 1; i < codigosPractium3.size(); i++)
        listaCodigosPractium3=listaCodigosPractium3 + " OR " + "CodigoPractium = '" + codigosPractium3.get(i) + "'";


    //ALUMNOS DE P3
    rs = cmd.executeQuery("SELECT * FROM alumno WHERE Asignado='N' AND (" + listaCodigosPractium3 + ") ORDER BY Nota DESC");
        while (rs.next())
            if (!adjudicarAlumno(rs.getString("DNI"))) {
                alumnosSinAsignar.add(new Alumno(rs.getString("DNI"), rs.getString("CodigoPractium"), rs.getString("CodigoGrado"),
                        rs.getString("Nombre"), rs.getString("Nota")));
            }

        lec.generaXlsx(dirCarpeta,"", "", "", "PIII", "",alumnosSinAsignar,null,null,null,null,null,"sinAsignar");
        alumnosSinAsignar.clear();

}
consultaExitosa=false;
```

Fig. 66. Parte del método ordenarAlumnos() dedicada a los alumnos de PRACTIUM III.

Como puede observarse, primero se seleccionan todos los códigos de la tabla Practicum que tienen en su nombre la cadena de caracteres PRACTICUM III (esto se debe a que hay muchos Practicum III en la base de datos debido a las especialidades que existen). Una vez cogidos todos esos códigos de practicums y haberlos puesto en formato de consulta SQL, se realiza a la base de datos una consulta para que devuelva todos los alumnos matriculados en alguno de esos códigos de Practicums, y además se le exige a dicha consulta que devuelva los alumnos ordenados por nota media en orden descendente. Una vez que tenemos todos los alumnos, mandamos a cada uno de esos alumnos a ADJUDICAR ALUMNO, para comprobar si alguna de las elecciones que hizo ese alumno en el archivo ELECCIONES ALUMNOS es posible que sea asignada en función de las ofertas de los centros. Como se puede observar, a la hora de llamar a adjudicarAlumno() por cada uno de los alumnos, éste devolverá una

respuesta booleana. Si dicha respuesta es afirmativa quiere decir que el alumno ha sido asignado a alguna de las elecciones que solicitó. En cambio, si devuelve false quiere decir que o dicho alumno no ha solicitado nada (no hay ninguna entrada en la tabla FORMULARIO para dicho alumno), o bien porque las elecciones que marcó en su solicitud ya han sido escogidas antes por otros alumnos del sistema y no ha podido escoger ninguna. En este último caso se irán añadiendo dichos alumnos sin asignar a un ArrayList, y al finalizar la ejecución del programa será generado un Excel con esos alumnos que no han podido ser asignados para gestionarlos de forma manual. Esta asignación manual se realizará en la aplicación por medio de un archivo ".csv" en el que se indica en cada línea el DNI del alumno, y el código de la oferta asignar (esa oferta deberá ser una de las que no están asignadas en ese momento y que puede consultarse en el archivo ofertasSinAsignar.xlsx.).

En enlace con lo anterior, comentaremos ahora **ADJUDICAR ALUMNOS**. Este método recibe un alumno concreto, por parte de ORDENAR ALUMNOS, recibiendo un único parámetro (DNI). Con base al DNI, el programa buscará en la tabla formulario a ver si existe alguna/s solicitud/es con ese DNI. En caso de que no exista terminaría la ejecución del programa, pero si por el contrario ese alumno ha marcado alguna posible elección, el programa procederá a asignar. Ahora lo que hay que comprobar es si la elección número 1 de dicho alumno existe, esto es, coger los datos de la elección número 1 que son el centro que desea y la plaza que desea, y comprobar en la tabla OFERTA, si hay alguna tupla que cumple esas características. En caso de haber consultado la tabla OFERTA y haber encontrado la plaza que el alumno desea y en el centro que el alumno desea, esa elección número 1 será asignada, creándose una entrada en la tabla AIGNACIÓN y marcando esa oferta del centro con flag ASIGNADA='S'. En caso de no haber encontrado ninguna entrada en la tabla oferta con el centro y la plaza que el alumno quería, se pasaría a la segunda elección (elección nº2) y se repetirá el proceso anterior. El proceso se repite hasta que el alumno encuentre una plaza y un centro de alguna de sus elecciones (devolviendo TRUE en ese caso), o hasta que el alumno se haya quedado sin elecciones puesto que ninguna de las que él marcó ha podido ser asignada (devolviendo FALSE). En el siguiente diagrama se muestra una ligera representación de cómo sería el proceso:

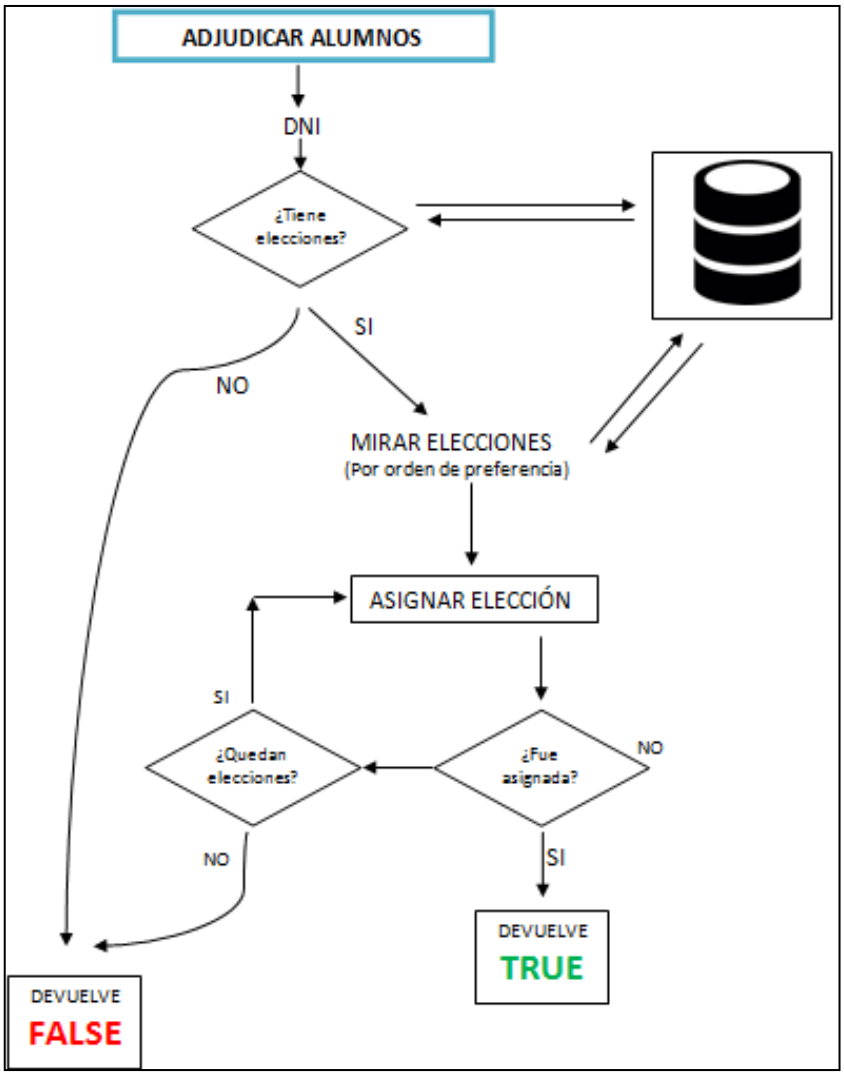


Fig. 67. Diagrama representativo de la asignación de alumnos a ofertas que solicitaron en su lista de elecciones.

En el anterior proceso cabe mencionar que los alumnos que marcaron en su elección Educación Especial, se les buscará todas las ofertas que sean de Pedagogía Terapéutica y de Audición y Lenguaje en el centro que marcaron, y se les será asignada una de las dos de forma totalmente aleatoria. Pues bien, con este último método terminaríamos la parte de asignación entre OFERTA y ALUMNOS, ahora procederemos a desarrollar la parte en la que se realiza la asignación entre PROFESOR y cada una de las asignaciones realizadas anteriormente.

La asignación de los profesores, como mencionamos en apartados anteriores, se puede realizar en base a dos criterios diferentes. En el primero de esos criterios los primeros profesores en seleccionar ofertas serán aquellos que tenga una mayor suma de PI, PII y PIII, es decir, los que mayor cantidad de ofertas se hayan ofrecido a tutorizar en total. El segundo de esos criterios se refiere a un orden de prioridad ya preestablecido que el usuario debe indicar en el fichero Excel con la información de los tutores.

Antes de empezar a desarrollar los algoritmos de selección de los profesores, mencionar que al iniciar la aplicación, antes de iniciar la ejecución del algoritmo, se le pedirá al usuario que escoja una letra de preferencia. En base a esa letra de preferencia serán ordenados más tarde los profesores por su distancia de la primera letra de su apellido respecto dicha letra. Esta letra de preferencia solo será utilizada en el primero de los métodos (el algoritmo que atiende al número de practicums en total a tutorizar). Por ejemplo, si ordenamos los profesores de mayor a menor número de Practicums a tutorizar, habrá algún profesor con 30 practicums a tutorizar, después otro con 24, después otro con 17, etc. Sin embargo por ejemplo a medida que vamos disminuyendo, nos encontraremos que habrá varios profesores que quieran tutorizar un total de 11 practicums, 10 practicums, etc.

Así pues, en esos casos en los que hay más de un profesor con un mismo número de Practicums a tutorizar, es necesario establecer algún otro criterio para que ese grupo de profesores sea ordenado.

De esta forma, si por ejemplo la letra de preferencia escogida fue la "F", dentro del grupo de profesores que quieran tutorizar un total de 10 Practicums, el orden de elección sería:

1. **F**ernández Soria, José. (Distancia = 0)
2. **G**arcía Rodríguez, Fernando. (Distancia = 1)
3. **Y**unque Gómez, David. . (Distancia = 19)
4. **E**steban Barrena, Manuel. (Distancia = 25)

Así pues, los profesores serán ordenados primero en base a las suma de practicums, y dentro de esta ordenación a su vez se realizará otra en base al atributo distancia, que se almacena en una columna en la tabla "Profesor" en el momento en que estos son cargados en la base de datos.

Puede existir el caso de que varios profesores no solo quieran tutorizar la misma cantidad de Practicum sino que también su atributo "Distancia" sea el mismo. En este caso los profesores serán ordenados por orden alfabético en base a su segunda letra, luego a la tercera, etc.

Así pues, en caso de empate la ordenación de los profesores siempre se realizará de forma alfabética en base a su apellido por lo que la única razón de ser del atributo "Distancia" es la posibilidad del usuario de elegir una letra de preferencia al comienzo.

En primer lugar explicaremos **ASIGNAR PROFESORES 1** o también llamado **ALGORITMO DE SUMA**. Este método ordena de mayor a menor los profesores en base al número total de Practicums a tutorizar, y dentro de esa ordenación, los profesores que quieren tutorizar la misma cantidad de Practicums, serán reordenados a su vez en base al atributo "Distancia" que hemos mencionado en el párrafo anterior.

Lo más eficiente a la hora de realizar el algoritmo sería que cada profesor tenga que hacer el mínimo número de desplazamientos posibles, es decir, que todas sus ofertas tutorizadas se encuentren en el mismo centro.

Es precisamente esa la razón por la que lo primero que haremos será saber si ese profesor "cabe" en alguno de los centros de la ZONA 1, es decir, si alguno de los centros de la ZONA 1 tiene las suficientes ofertas para tutorizar LIBRES que el profesor ha solicitado impartir.

Si efectivamente hay algún centro en la Zona 1 con esa cantidad de ofertas, será asignado y se pasará al siguiente profesor. Sin embargo, si no ha podido encontrarse ningún centro en la Zona 1, se pasará a la siguiente zona.

En caso de no haber podido ser asignado, se realizará el mismo procedimiento pero ahora se buscarán centros en la ZONA 2, es decir, se buscará si en la totalidad de los centros de la ZONA 2 hay alguno que tiene suficientes plazas, y será asignado. Si no ha sido posible tampoco, entonces se realizará el mismo procedimiento con la ZONA 3 y con la ZONA 4 (el programa no tiene limitación en este aspecto, por lo que si fuesen 9 zonas en lugar de 4 sería exactamente igual pero realizando comprobaciones hasta la ZONA 9).

Si en la totalidad de las zonas y en la totalidad de sus centros no ha cabido enteramente en ningún centro, entonces se pasará al método "**CABE EN ZONAS**".

Este método se explicará posteriormente y su función es que ya que no se ha podido asignar a un profesor a un solo centro, que éste sea asignado al menor número de centros, por lo que este método se encarga de realizar el reparto de las ofertas a tutorizar en el menor número de centros posibles de las zonas que los profesores eligieron. Si ha sido posible repartirlo en diferentes centros, ese profesor quedará asignado, mientras que si ni siquiera repartiéndolo en diversos centros ha sido posible llevar a cabo la asignación, entonces se detendrá el algoritmo, se almacenarán en la base de datos todos las asignaciones que se han hecho hasta ese momento, y se solicitará al usuario que inserte manualmente las asignaciones de ese profesor "problemático" si se quiere proseguir con la ejecución, o terminarla y mostrar los resultados si no se quiere continuar.

```
rs = cmd.executeQuery("SELECT * FROM profesor WHERE Asignado='N' ORDER BY Suma DESC");
while (rs.next() && !terminada) {
    rs2 = cmd2.executeQuery("SELECT * FROM profesor WHERE Asignado='N' AND Suma='" +
            rs.getString("Suma") + "' ORDER BY Distancia ASC");

    while (rs2.next() && !listaDNIAAsignados.contains(rs2.getString("DNIprofesor")) && !terminada) {
            cabe=false;
            listaZonas=formatoZonas(rs2.getString("Zonas"));

            for (int i = 0; i < listaZonas.size(); i++) {
                CodigoCabeEnCentro=cabeEnCentro(listaZonas.get(i), rs2.getString("P1"), rs2.getString("P2"),
                        rs2.getString("P3"), rs2.getString("P3I"), rs2.getString("P3P"));
                if (!CodigoCabeEnCentro.equals("")) {
                    asignarProfesorManual(rs2.getString("DNIprofesor"),  cabeEnCentro(listaZonas.get(i), rs2.getString("P1"),
                            rs2.getString("P2"), rs2.getString("P3"), rs2.getString("P3I"), rs2.getString("P3P")));
                    cabe=true;
                    listaDNIAAsignados.add(rs2.getString("DNIprofesor"));
                    i=listaZonas.size();
                }
            }

            if (!cabe)
                if(!cabeEnZonas(rs2.getString("DNIprofesor"),listaZonas, rs2.getString("P1"), rs2.getString("P2"),
                        rs2.getString("P3"), rs2.getString("P3I"), rs2.getString("P3P"))) {
                    //exportaManual
                    terminada=true;
                    devuelto= rs2.getString("DNIprofesor");

                } else
                    //Asignacion realizada correctamente
                    listaDNIAAsignados.add(rs2.getString("DNIprofesor"));
        }
    }

conexion.commit();
return devuelto;
```

Fig. 68. Código del método **asignarProfesores1()**.

En la siguiente imagen se muestra el diagrama correspondiente a lo que se acaba de explicar y a lo que hace referencia la sección de código anterior. Cabe mencionar, que como ya se dijo anteriormente, aunque en el diagrama se han representado 4 zonas, esta variable no es fija, y puede utilizarse un número ilimitado. La razón por la que en el ejemplo se han puesto 4 es porque es el criterio utilizado en cursos anteriores.

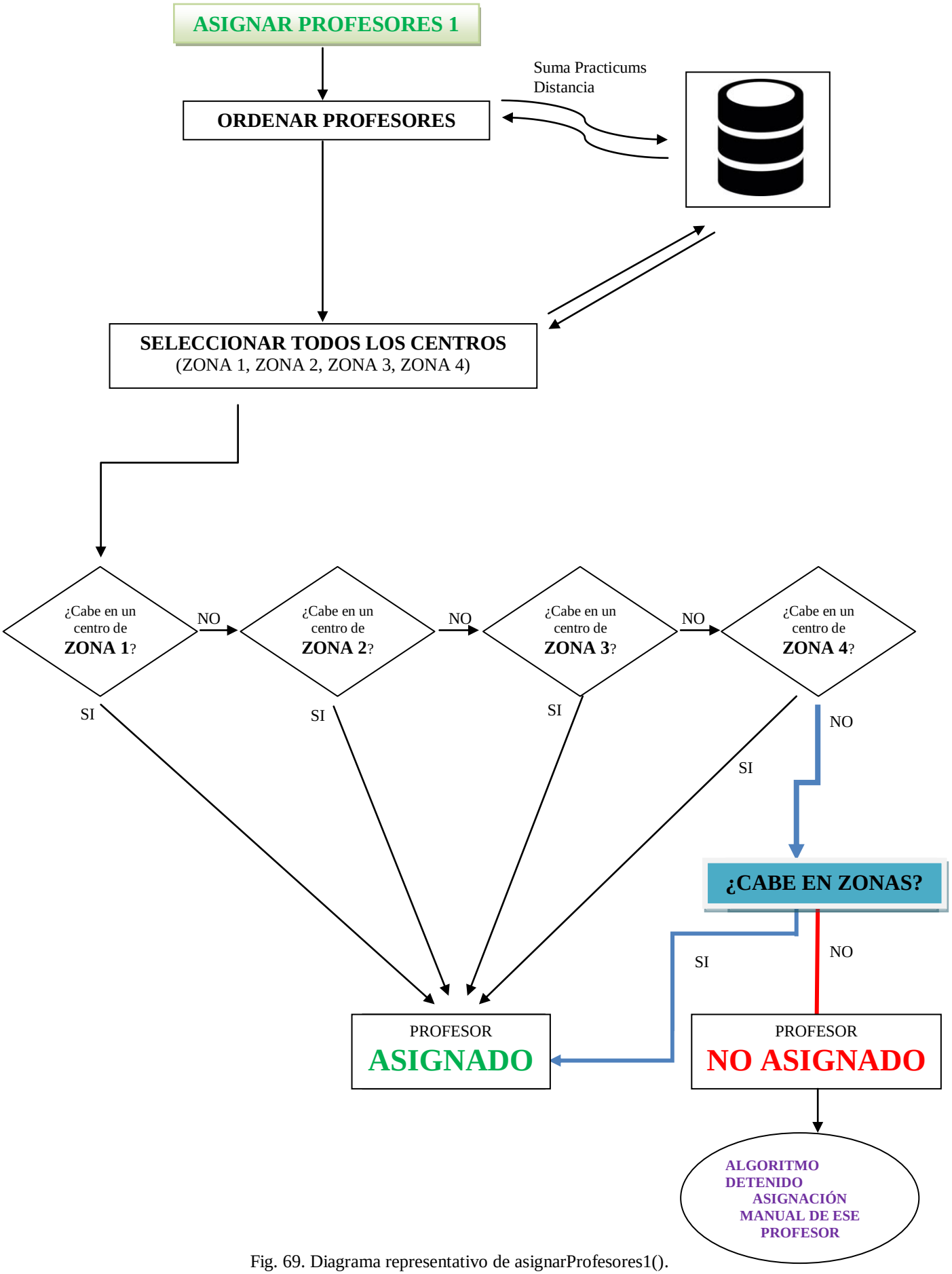


Fig. 69. Diagrama representativo de asignarProfesores1().

En segundo lugar explicaremos la otra posibilidad de asignar profesores, que es el método **ASIGNAR PROFESORES 2** o también llamado **ALGORITMO DE PRIORIDAD.** Este método realiza exactamente lo mismo que el método anterior, con la única diferencia de que en el primero de los pasos, a la hora de ordenar los profesores, no los ordena por el método de practicums anterior, sino que los ordena en base a una columna llamada "Prioridad". Así pues, el primer profesor en elegir plaza será aquel que tenga Prioridad="1", después el que tenga Prioridad="2", etc.

Es decir, en este método, los profesores ya siguen un orden concreto que ha sido establecido por el usuario a la hora de insertar el Excel.

Como detalle cabe mencionar que el método a elegir para realizar la asignación es el ALGORITMO DE SUMA, apareciendo el ALGORITMO DE PRIORIDAD como método secundario en caso de que la facultad, por alguna razón, quiera realizar la asignación de profesores con un orden a su propio criterio.

Como hemos dicho, lo único que cambia respecto al método de ALGORITMO DE SUMA es el orden de los profesores, ya que el procedimiento de cabe en centro, cabe en zonas, etc. es exactamente el mismo con el pequeño detalle de que ahora la distancia respecto de la letra de referencia establecida por el usuario no será tenida en cuenta, ya que no habrá casos de "empates" de profesores.

```
//Los ordenamos solo con orden de Prioridad
rs = cmd.executeQuery("SELECT * FROM profesor WHERE Asignado='N' ORDER BY Prioridad ASC");

while (rs.next() && !terminada) {

            cabe=false;
            listaZonas=formatoZonas(rs.getString("Zonas"));

            for (int i = 0; i < listaZonas.size(); i++) {
                CodigoCabeEnCentro=cabeEnCentro(listaZonas.get(i), rs.getString("P1"), rs.getString("P2"),
                        rs.getString("P3"), rs.getString("P3I"), rs.getString("P3P"));
                if (!CodigoCabeEnCentro.equals("")) {
                    asignarProfesorManual(rs.getString("DNIprofesor"),  cabeEnCentro(listaZonas.get(i), rs.getString("P1"),
                            rs.getString("P2"), rs.getString("P3"), rs.getString("P3I"), rs.getString("P3P")));
                    cabe=true;
                    i=listaZonas.size();
                }
            }

            if (!cabe)
                if(!cabeEnZonas(rs.getString("DNIprofesor"),listaZonas, rs.getString("P1"), rs.getString("P2"),
                        rs.getString("P3"), rs.getString("P3I"), rs.getString("P3P"))) {
                    //exportaManual
                    terminada=true;
                    devuelto= rs.getString("DNIprofesor");
                } else
                    System.out.println("Asignacion distribuida");
            }
```

Fig. 70. Código del método **asignarProfesores2().**

El diagrama del ALGORITMO DE PRIORIDAD es el mismo que aparece en la **Figura 69,** con la única diferencia de que ahora no se ordena por "Suma de Practicums" y por "Distancia", sino por "Prioridad".

Ahora explicaremos el funcionamiento de CABE EN CENTRO y de CABE EN ZONAS, que son llamados por los dos métodos vistos anteriormente, y cuya explicación es relevante de cara a entender el funcionamiento del algoritmo.

CABE EN CENTRO es usado tanto por asignarProfesores1 () como por asignarProfesores2 () para la primera de las comprobaciones que hemos mencionado, que es la de ver si un profesor cabe en un solo centro, es decir, saber si todas las ofertas que quiere tutorizar es posible que se impartan en el mismo centro.

Así pues, una vez que hemos ordenados todos los profesores, en base a suma de Practicums o en base a Prioridad (dependiendo del algoritmo elegido), lo primero que hacemos es llamar a esta función. Para cada profesor, esta función será llamada tantas veces como zonas tenga, es decir, si por ejemplo el profesor tiene 4 zonas de preferencia, esta función será llamada 4 veces.

Para cada una de ellas llamadas, CABE EN CENTRO recibirá como parámetro la zona que se quiere analizar y las ofertas que quiere tutorizar, y se encargará de buscar si en la zona indicada hay algún centro que tenga tantas ofertas como el profesor quiere tutorizar.

En caso de que este método haya encontrado algún centro con esas características, entonces devolverá el valor del código de ese centro, mientras que si no ha encontrado ningún centro en esa zona, devolver una cadena vacía.

Fig. 71. Código del método cabeEnCentro().

Como puede verse, primero se extraen TODOS los colegios de la zona indicada y después para cada uno de esos centros se realizan las siguientes comprobaciones:

- Comprobar si en el centro concreto hay suficientes ofertas de PI, es decir, si hay suficientes ofertas en ese colegio con código de Practicum '420012', '430006' o '421009' (códigos correspondientes al Practicum I).
- Si hay suficientes ofertas de PI, entonces ahora se comprobará si hay suficientes ofertas de PII, es decir, si hay suficientes ofertas en ese colegio con código de Practicum '420023' o '430018' (códigos correspondientes al Practicum II).
- Si también ha habido suficientes ofertas de PII, ahora por último se comprobará si existen suficientes ofertas de PIII. Para saber si hay suficientes ofertas de PIII, nos valdrá con que esas ofertas sean de alguno de los Practicums específicos indicados por el tutor (internamente denominados 'P3I' y 'P3P'), o bien que sean de Practicums generalistas (con códigos '420027' y '430020' y cuyo funcionamiento ya se explicó anteriormente). Es decir, si por ejemplo los códigos de practicums específicos de un tutor son '420037' y '430454', se tratará de averiguar si hay suficientes ofertas en ese colegio con códigos de Practicum '420037', '430454', '420027' o '430020' (específicos más generalistas).

En caso de que se haya demostrado que ese centro tiene espacio para albergar los PI, PII y PIII, entonces se devolverá el código de ese centro en concreto, mientras que si no ha "cabido" en ese centro, se pasará a comprobar el siguiente centro en la lista de esa zona. Si finalmente se han recorrido todos los centros de la zona indicada y ninguno tiene la capacidad para albergar se devolverá cadena vacía.

Se realizará el mismo procedimiento de llamada a CABE EN CENTRO tantas veces como zonas tenga el profesor, como se indica en la **Figura 69**.

Si llegados a este punto, en ninguna de las 4 zonas indicadas por el profesor ha habido ni un solo centro en el que ese profesor haya podido ser asignado íntegramente, entonces se pasará al método **CABE EN ZONAS**.

Ya que un profesor no ha podido ser asignado a un solo centro, este método se encarga de repartir el profesor en cuestión al menor número de centros posibles en las zonas indicadas.

Cabe mencionar que al contrario que CABE EN CENTRO, ahora la ZONA 1 no es la preferente sino que no hay un orden de preferencia de zonas establecido. En este método todas las zonas tienen la misma importancia, y al realizar la búsqueda lo que se hace es buscar en la totalidad de los centros de la todas las zonas indicadas, y repartir ese profesor en el menor número de centros posibles, sin tener en cuenta si uno de esos centros por ejemplo es de la zona 1 o de la zona 3, otro de la zona 2, etc.

Como hemos mencionado, una vez capturados todos los centros de la totalidad de las zonas indicadas se tratará de repartir ese profesor en el menor número de centros posibles, es decir, de escoger primero los centros con mayor cantidad de ofertas sin tutorizar. Para ello es necesario ordenar los centros de mayor a menos número de ofertas que necesitan ser tutorizadas, y para ello a su vez fue necesario

añadir una columna a la tabla "Asignación" que nos sirva de referencia para realizar esa ordenación.

La columna que precisamente nos permite realizar la ordenación es '**SumaOfertasCentro'**. Se trata de un valor numérico que para cada asignación nos indica el NÚMERO de ASIGNACIONES SIN TUTORIZAR de ese mismo tipo de Practicum en ese MISMO CENTRO.

En la siguiente imagen que utilizamos a modo de ejemplo se muestran solo las asignaciones de un determinado centro elegido al azar, que en este caso es el '93'.

DNI	CodigoPractium	CodigoCentro ▾ 1	CodigoOferta	Tutorizada	SumaOfertasCentro
50998893	420046	93	740	N	003
47232205	420012	93	744	N	004
02591997	420012	93	743	N	004
49390998	420012	93	742	N	004
02716032	430053	93	739	N	004
52893692	430053	93	738	N	004
52011495	420027	93	748	N	002
49147238	420027	93	741	N	002
52018794	420044	93	747	N	003
47234621	430006	93	746	N	004

Fig. 72. Asignaciones sin tutorizar del centro con código '93'.

Para analizar el parámetro SumaOfertasCentro, vamos a dividir las asignaciones del centro '93' en tres tipos.

- En primer lugar tenemos las asignaciones con **PI**.

  Recordemos de apartados anteriores que PI son aquellas asignaciones que tienen bien CodigoPractium='420012', bien CodigoPractium='430006' o bien CodigoPractium='421009'.

  El profesor que haya indicado por ejemplo querer tutorizar cuatro PRACTICUM I, le va a ser indiferente que esos practicum I tengan código '420012', '430006' o '421009'. Es por esta razón por la que en

SumaOfertasCentro aparecen para cada asignación con PRACTICUM I, la suma del total de PRACTICUM I que hay en ese centro. Como puede verse en la **Figura 72**, la segunda, la tercera, la cuarta y la última línea, aparece SumaOfertasCentro='4', ya que es ese el número de PRACTICUM I en ese centro.

- En segundo lugar tenemos las asignaciones con **PII**.
  Este caso es similar al anterior, solo que ahora los códigos son '420023' y '430018'. En el centro expuesto en la **Figura 72** no aparece ninguna asignación con este código de Practicum pero su funcionamiento sería igual que el anterior, es decir, los profesores que quieran tutorizar por ejemplo ocho PII, les da igual que esas ocho asignaciones tengan cualquiera de estos dos últimos códigos, por lo que en SumaOfertasCentro aparecerá la suma total de asignaciones con alguno de esos dos practicums en ese mismo centro.

- En último lugar tenemos las asignaciones con **PIII**.
  Este caso es más complejo que los anteriores. Los profesores que hayan indicado en su especialidad Practicums generalistas, solo podrán impartir Practicums generalistas ('420027' y '430020'). Sin embargo los profesores que hayan indicado en su especialidad otros dos tipos de Practicums que no sean generalistas como por ejemplo '420059' y '430035', podrán impartir en total cuatro, es decir, estos dos últimos más los dos generalistas.
  Es por esta razón, por la que en la **Figura 72** las asignaciones con Practicums generalistas como la siete y la ocho que tienen CodigoPractium='420027', su valor de SumaOfertasCentro='2', ya que en ese centro solo hay dos generalistas.
  - *Esto lo que quiere decir es que los profesores que indicaron en su especialidad los códigos generalistas, solo pueden tutorizar dos plazas en este centro.*

Sin embargo, en el caso de la primera asignación de la lista que tiene CodigoPractium='420046', su valor SumaOfertasCentro='3' ya que

estamos contando esa asignación con CodigoPractium=’420046’ y las dos generalistas, lo que hace una suma de 3 en total.

- *Esto lo que quiere decir es que los profesores que indicaron en su especialidad el código ‘420046’ en este centro tendrán tres posibles asignaciones que tutorizar que son la de la primera línea, la séptima y la octava.*

El caso de la penúltima asignación con CodigoPractium=’420044’ es exactamente idéntico al que acabamos de explicar.

Así pues, cuando llamemos al método CABE EN ZONAS, este realizará una búsqueda en las asignaciones de todos los centros de la totalidad de las zonas que el profesor indicó, y las ordenará de mayor a menor SumaOfertasCentro, lo que permitirá que el profesor en cuestión sea repartido

Cabe mencionar que a medida que un profesor vaya “cogiendo” asignaciones dentro del algoritmo, se buscará si hay algún centro en esas zonas que pueda albergar la totalidad de las asignaciones que le quedan a ese profesor por tutorizar.

Ejemplo: Si un profesor quiere tutorizar veinticuatro PRACTICUM I, se ordenará de mayor a menor las asignaciones tal y como hemos dicho anteriormente, de forma que quedarían así:

- 9 asignaciones de PRACTICUM I en centro 10. (Quedan 15 por asignar)
- 7 asignaciones de PRACTICUM I en centro 87. (Quedan 8 por asignar)
- 6 asignaciones de PRACTICUM I en centro 24. (Quedan 2 por asignar)
- 6 asignaciones de PRACTICUM I en centro 59.

…

Pues bien, primero el profesor cogería las 9, luego las 7, luego las 6, y por último cogería las dos que le quedan del último señalado en naranja. Sin embargo esto genera un grave problema a posteriori ya que es posible que un profesor que quiera

tutorizar 6 asignaciones, ya no pueda coger las del centro 59, porque alguien ha cogido dos de ellas y ahora solo quedan 4.

Esta forma de realizar la asignación que ahora parece no tener importancia, en los momentos finales del algoritmo causa que las asignaciones que queden sean todas con SumaOfertasCentro='3', SumaOfertasCentro='2', etc., lo que causa que los últimos profesores sean asignados a muchísimos centros diferentes y tengan que realizar mayor cantidad de desplazamientos en el curso académico.

Para evitar este problema, a medida que el algoritmo va realizando asignaciones entre profesores y ofertas, COMPRUEBA si en ALGUNO DE LOS CENTROS de ALGUNA DE LAS ZONAS hay un centro que tiene exactamente las asignaciones que quedan a tutorizar por ese profesor.

En el ejemplo expuesto se ejecutarían las 22 primeras asignaciones, y cuando quedasen solo dos ofertas por tutorizar para ese profesor, antes de coger las ofertas del centro naranja (que sería el último remedio), buscaría si hay ALGÚN CENTRO en las zonas indicadas por el profesor que tenga exactamente DOS OFERTAS sin tutorizar, y las asignaría a ese profesor.

Si por ejemplo ha encontrado el centro='65' que tiene dos ofertas de PRACTICUM I sin tutorizar, las últimas dos asignaciones del profesor serían en ese centro. Este último paso causaría que el centro='65' ya haya sido "llenado" en cuanto a los PRACTICUM I, y además que el centro 59 siga teniendo seis plazas de PRACTICUM I de cara a futuras asignaciones, lo que supone una gran ventaja.

En la siguiente imagen se muestra un fragmento de código del método CABE EN ZONAS, concretamente el referido a las asignaciones de PRACTICUM I, en el que se realiza lo anteriormente expuesto. En el programa, debajo de la siguiente imagen aparecería un fragmento de código similar pero para PRACTICUM II y otro para el caso de PRACTICUM III, con la única diferencia de los Códigos de Practicum como hemos dicho anteriormente.

```
//Ahora ya hemos comprobado que todo cabe en esas zonas, así que procedemos a la asignación
boolean terminado=false;


//P1
rs = cmd.executeQuery("SELECT * FROM asignacion WHERE Tutorizada='N' AND (" + listaCentrosSQL + ") AND (CodigoPractium='420012'"
        + " OR CodigoPractium='430006' OR CodigoPractium='421009') ORDER BY SumaOfertasCentro DESC, CodigoCentro DESC" );

while (rs.next() && PE1>0 && !terminado) {
    actualizarSumasDeOfertas(listaCentrosSQL);

    rs2 = cmd2.executeQuery("SELECT * FROM asignacion WHERE SumaOfertasCentro='" + formatoEnteroBaseDeDatos(String.valueOf(PE1)) +
            "' AND Tutorizada='N' AND (" + listaCentrosSQL + ") AND (CodigoPractium='420012' OR CodigoPractium='430006' OR "
            + "CodigoPractium='421009') ORDER BY SumaOfertasCentro DESC, CodigoCentro DESC" );

    while (rs2.next() && PE1>0) {
            PreparedStatement insertAsignacionProfesor=obtenerInsertAsignacionProfesor(rs2.getString("DNI"),
                    rs2.getString("CodigoPractium"), rs2.getString("CodigoCentro"), rs2.getString("CodigoOferta"), DNIprofesor);
            insertAsignacionProfesor.executeUpdate();
            PreparedStatement cambiarFlag1=cambiarEstadoAsignacion(rs2.getString("CodigoOferta"));
            cambiarFlag1.executeUpdate();
            PE1--;
            terminado=true;
    }

    if (!terminado) {
            PreparedStatement insertAsignacionProfesor=obtenerInsertAsignacionProfesor(rs.getString("DNI"),
                    rs.getString("CodigoPractium"), rs.getString("CodigoCentro"), rs.getString("CodigoOferta"), DNIprofesor);
            insertAsignacionProfesor.executeUpdate();

            PreparedStatement cambiarFlag1=cambiarEstadoAsignacion(rs.getString("CodigoOferta"));
            cambiarFlag1.executeUpdate();

            PE1--;
        }
}
```

Fig. 73. Fragmento de código de CABE EN ZONAS referido a las asignaciones de PRACTICUM I.

En caso de que el profesor haya podido ser asignado mediante CABE EN ZONAS, se modificara el estado de ese profesor con Asignado=’S’ y se pasará a realizar la asignación del siguiente profesor en la lista. Sin embargo, si ni siquiera repartiéndolo en centros diferentes ha sido posible llevar a cabo su asignación, entonces se exportarán los datos de las asignaciones realizadas hasta ese momento y se solicitará al usuario que ese profesor “problemático” sea introducido de forma manual.

En este momento el usuario puede bien terminar la ejecución del programa y analizar los ficheros resultantes hasta ese momento, o bien introducir manualmente ese profesor “problemático” indicando la lista de asignaciones que va a tutorizar y continuar después con la ejecución del algoritmo desde donde se quedó la última vez, es decir, analizaría el SIGUIENTE profesor al “problemático” que se encuentre en la lista.

5.2.5. Start.java

En esta clase se realiza la ejecución del programa, es donde se encuentra el Main(). A la hora de la verdad esta clase no va a ser ejecutada, puesto que no tiene interfaz, solo líneas de código, y el usuario lo que ejecutará será el main presente en otra clase, que es la clase PRINCIPAL dentro de la carpeta INTERFAZ como se comentará más adelante. Esta clase por tanto ha sido desarrollada para realizar las pruebas oportunas de la programación y para comprobar que toda la gestión, asignaciones, y demás tareas del programa las realiza correctamente antes de ponerse a programar la interfaz.

Así pues, aunque no va a ser ejecutada en un principio de cara al usuario, he creído conveniente conservarla por si en algún momento durante la vida del presente proyecto es necesario probar determinadas funciones de la aplicación que no puedan ser llamadas desde la interfaz, de cara a la detección y depuración de errores, conexiones a la base de datos, modificaciones de funciones, etc.

Actualmente esta clase puede realizar y realiza las mismas llamadas que la interfaz gráfica de cara al usuario.

```
public class Start {


    public static void main(String[] args) throws IOException {

                        ModeloBBDD mod = new ModeloBBDD();
                        LecturasFicheros lec= new LecturasFicheros();

                        //mod.ordenarAlumnos();

                        lec.leerCentrosGuadalajara(mod);
                        lec.leerCentrosMadrid(mod);

                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas1.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas2.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas3.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas4.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas5.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas6.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas7.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas8.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas9.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas10.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas11.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas12.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas13.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas14.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas15.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas16.csv");
                        lec.leerMatriculas(mod, "C:/Users/Victorino/Desktop/tfg/Practicum_NotaMedia/matriculas17.csv");

                        lec.leerFormulario(mod, "C:/Users/Victorino/Desktop/tfg/Eleccion_Alumnos/formulario01.csv");
                        lec.leerFormulario(mod, "C:/Users/Victorino/Desktop/tfg/Eleccion_Alumnos/formulario2.csv");
                        lec.leerFormulario(mod, "C:/Users/Victorino/Desktop/tfg/Eleccion_Alumnos/formulario3.csv");
                        lec.leerFormulario(mod, "C:/Users/Victorino/Desktop/tfg/Eleccion_Alumnos/formulario4.csv");
                        lec.leerFormulario(mod, "C:/Users/Victorino/Desktop/tfg/Eleccion_Alumnos/formulario5.csv");
                        lec.leerFormulario(mod, "C:/Users/Victorino/Desktop/tfg/Eleccion_Alumnos/formulario6.csv");
```

Fig. 74. Fragmento de código de la clase Start().

5.2.6. Asignación.java

Representa las ofertas que ya están asignadas a un determinado alumno, pero que no tienen asignado ningún profesor de momento. Es decir, aquellas asignaciones ALUMNO-OFERTA que todavía no están tutorizadas por nadie.

La razón de ser de esta entidad es simplificar el envío de datos al método generarXlsx(), ya que esto nos permite agrupar en un ArrayList todas esas ofertas, para que más tarde ese método pueda generar el archivo **ofertasSinTutorizar.xlsx.**

Nota: Cuando queramos realizar una asignación manual entre profesores y ofertas, debemos indicar el profesor en cuestión y una lista de ofertas sin tutorizar, es decir, solo se podrán elegir ofertas de la lista proporcionada en el archivo en cuestión.

5.2.7. AsignaciónFinal.java

Representa las ofertas que ya están asignadas a un determinado alumno, y que también han sido ya tutorizadas por un determinado profesor. Es decir, representa el resultado final de la aplicación, puesto que aparecen asignaciones ALUMNO-OFERTA-PROFESOR.

La razón de ser de esta entidad es simplificar el envío de datos al método generarXlsx(), ya que esto nos permite agrupar en un ArrayList todas esas ofertas, para que más tarde ese método pueda generar el archivo **ResultadoFinal.xlsx.**

5.2.8. Oferta.java

Representa las ofertas que no están asignadas de momento a ningún alumno, es decir, ofertas que han publicado los centros pero que todavía ningún alumno ha solicitado impartirla.

La razón de ser de esta entidad es simplificar el envío de datos al método generarXlsx(), ya que esto nos permite agrupar en un ArrayList todas esas ofertas, para que más tarde ese método pueda generar el archivo **ofertasSinAsignar.xlsx.**

Nota: Cuando queramos realizar una asignación manual entre una oferta y un alumno, es decir, realizar la asignación manual ALUMNO-OFERTA, debemos indicar el alumno en cuestión y la oferta que se le va asignar. Esa oferta a asignar debe de ser una de las que aparecen en la lista del archivo ya mencionado, puesto que son las que están libres en ese momento.

5.2.9. OfertaAsignada.java

Esta clase guarda especial relación con la ya comentada "**Asignación.java**". Cada vez que la aplicación genera o modifica el archivo **ofertasSinTutorizar.xlsx** se generará una copia con la MISMA INFORMACIÓN pero en formato ".csv" que se llamará "**OfertasAsignadasModoManual.csv**".

Así pues, **ofertasSinTutorizar.xlsx** aparece en un formato mucho más legible y más fácil de interpretar de cara a que el usuario de la aplicación pueda consultar la información de forma rápida, puesto que aparece información como el DNI del Alumno, el código de la oferta, el código del Practicum, su nombre, la localidad, etc.

En cambio **OfertasAsignadasModoManual.csv** muestra la misma información, pero indicando solo dos columnas que son el DNI del Alumno y el CÓDIGO DE OFERTA asignada.

La utilidad de generar dos archivos con la MISMA INFORMACIÓN (recordemos que es la lista de ofertas que ya están asignadas a un alumno pero que no tienen profesor) en distinto formato radica en que **ofertasSinTutorizar.xlsx** es utilizado como un método de consulta de cara al usuario, bien para ver información acerca de las asignaciones realizadas hasta ese momento o bien para escoger de forma rápida y eficiente alguna de las ofertas que están sin tutorizar para realizar una asignación manual entre PROFESOR y OFERTA.

En cambio **OfertasAsignadasModoManual.csv** no se presenta como un archivo de consulta (aunque se puede consultar perfectamente, pero al estar en formato CSV es más difícil su interpretación de cara al usuario), sino será utilizado en el caso de que queramos detener la ejecución en un momento concreto y al volver a ejecutarla solo tengamos que seleccionar este archivo de asignación manual para

volver al mismo punto en el que nos quedamos anteriormente, sin necesidad de volver a correr el algoritmo para realizar otra vez las asignaciones.

Es decir, el formato en el que se almacena la información en **OfertasAsignadasModoManual.csv** coincide con el formato requerido realizar la asignación manual entre un alumno y una oferta.

Esta funcionalidad se añadió debido al requerimiento de ejecutar la aplicación en distintos momentos del curso académico.

5.3. ICONOS

Este paquete contiene las imágenes y los iconos que son utilizados de forma gráfica por la aplicación, y que deben ser almacenados también junto al resto del proyecto de cara a que cuando la aplicación sea ejecutada en otro ordenador la interfaz del programa no sufra modificaciones.

5.4. INTERFAZ

5.4.1. Principal.java

Esta entidad es la más importante del paquete interfaz. Es la que actúa como **MAIN**, es decir, es la entidad a partir de la cual empieza a correr la aplicación.

Nada más iniciar el programa es esta ventana la que aparece, actuando a modo de PANEL CENTRAL en el que se puede controlar todos los aspectos del proceso de asignación de alumnos y de profesores, tal y como podrá verse más adelante en el Manual de Usuario.

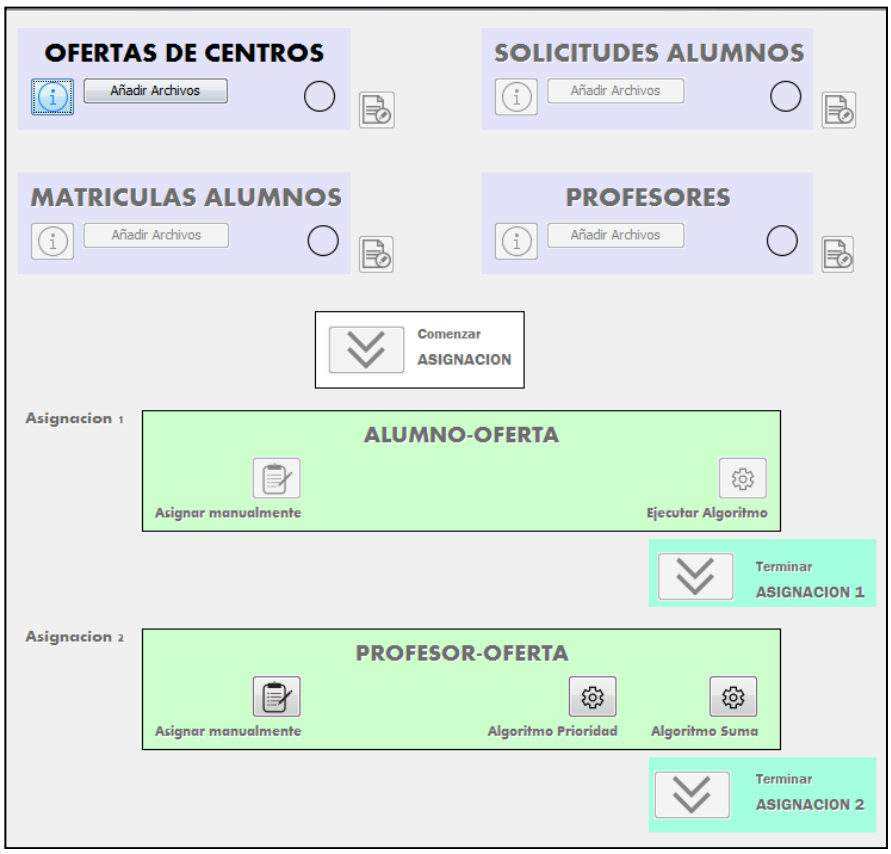


Fig. 75. Aspecto de la entidad Principal.java.

5.4.2. InterfazCentros.java

Esta entidad es a través de la cual se especifica las direcciones de los archivos en formato CSV donde se encuentra la información de los centros de Guadalajara y de los centros de Madrid, a partir de los cuales se generarán los centros y las ofertas en la base de datos.

Fig. 76. Aspecto de la entidad InterfazCentros.java.

5.4.3. InterfazMatrículas.java

Esta entidad es a través de la cual se especifica la lista de direcciones de los archivos CSV que conforman las matrículas de los alumnos con información acerca del practicum matriculado, nota media, etc.

Fig. 77. Aspecto de la entidad InterfazMatrículas.java.

5.4.4. InterfazProfesores.java

Esta entidad es a través de la cual se especifica la lista de direcciones de los archivos CSV que conforman la lista de los profesores con sus practicum a tutorizar y las zonas de preferencia.

Fig. 78. Aspecto de la entidad InterfazProfesores.java.

5.4.5. InterfazSolicitudes.java

Esta entidad es a través de la cual se especifica la lista de direcciones de los archivos CSV que conforman las elecciones o formularios de los alumnos, es decir, los archivos donde se encuentren N elecciones por orden de preferencia que han realizado los alumnos en cuanto a las plazas en las que desean ser asignados.

Fig. 79. Aspecto de la entidad InterfazSolicitudes.java.

5.4.6. InterfazDirectorioGeneral.java

Una vez que ya se han seleccionado los archivos de ALUMNOS, CENTROS, FORMULARIOS y PROFESORES, esta entidad sirve para especificar la letra de preferencia en base a la cual serán ordenados los profesores por el Algoritmo de SUMA y el directorio donde queremos que se guarden los resultados (Ficheros Excel). Al pulsar "Aceptar" comenzará la carga de los datos especificados en los ficheros Excel de entrada en la base de datos, por lo que podríamos decir que es una entidad que hace de FRONTERA entre la información contenida en los archivos Excel iniciales y la ejecución del Algoritmo.

Fig. 80. Aspecto de la entidad InterfazDirectorioGeneral.java.

5.4.7. InterfazManualAlumnos.java

Una vez que se ha realizado la carga masiva de datos en la base de datos, es cuando podemos empezar a trabajar en cuanto a aspectos de asignación se refiere. Esa asignación puede realizarse automáticamente mediante algoritmos por medio de los botones que el panel central ofrece al respecto, o puede realizarse manualmente.

Esta entidad permite la inserción manual de asignaciones ALUMNO-OFERTA, especificando en ella la ruta del archivo en formato CSV donde se encuentre la información acerca de las asignaciones manuales que se deseen realizar. El formato específico de ese archivo se indica en el manual de usuario.

Fig. 81. Aspecto de la entidad InterfazManualAlumnos.java.

5.4.8. InterfazManualProfesores.java

Una vez que hemos terminado la asignación de alumnos y ofertas bien por medio de la asignación manual de alumnos y ofertas, bien por medio del modo automático de asignación, o bien utilizando ambos métodos, pasamos a la parte de asignación entre OFERTAS y PROFESORES.

Al igual que el apartado anterior, esta entidad permite la inserción manual de asignaciones PROFESOR-OFERTAS, especificando en ella la ruta del archivo en formato CSV donde se encuentre la información acerca de las asignaciones manuales que se deseen realizar. El formato específico de ese archivo se indica en el manual de usuario.

Fig. 82. Aspecto de la entidad InterfazManualProfesores.java.

6. PRESUPUESTO

Esta aplicación no tiene fines comerciales y por tanto no tiene coste alguno, sino que nace como proyecto de fin de grado para dar solución a la necesidad de asignar profesores, alumnos y ofertas de forma mucho más rápida y eficiente. Sin embargo, en el presente apartado se ha realizado una estimación del coste que conllevaría el desarrollo del presente proyecto.

PERSONAL					
Tipo	**Nº**	**Precio/Hora**	**Tarea**	**Horas de trabajo**	**Total**
Analista	1	20 €/hora	Diseño de BBDD	40	800 €
Programador	1	15 €/hora	Leer ficheros	30	450 €
Programador	1	15 €/hora	Interacciones con base de datos	20	300 €
Programador	1	15 €/hora	Diseño algoritmo alumno-oferta	60	900 €
Programador	1	15 €/hora	Diseño algoritmo Profesor-oferta	55	825 €
Programador	1	15 €/hora	Generación de resultados	25	375 €
Diseñador	1	20 €/hora	Diseño de interfaz	30	600 €
TOTAL				**260 horas**	**4250 €**

MATERIALES			
Tipo	**Nº**	**Nombre**	**Precio**
Hardware	1	Ordenador	700 €
Software	1	Netbeans	0 €
Software	1	Paquete XAMPP	0 €
TOTAL			**700 €**

TOTAL	**4950 €**

Como puede observarse en las anteriores tablas, si la aplicación hubiera sido desarrollada con fines comerciales, hubiera tenido un coste de 4950 € sumando los gastos de programación y los gastos de materiales.

Sin embargo, aunque en las tablas anteriores únicamente se ha calculado el coste de desarrollo, la IMPLANTACIÓN REAL de esta aplicación llevaría asociado unos gastos de mantenimiento y de soporte de cara al comprador con el fin solventar futuros requerimientos, y cuyo coste vale la pena mencionar. Estos costes de mantenimiento se dividen en una parte fija que se paga anualmente, y en una parte variable en función de las horas de trabajo de mantenimiento.

PARTE FIJA		
Tipo	**Precio**	**Descripción**
Soporte Anual	300 €	Soporte anual por mantenimiento de la aplicación que el comprador debe abonar para poder resolver futuros requerimientos. Cuota a pagar por tener la posibilidad de contacto con la empresa.

PARTE VARIABLE					
Tipo	**Nº**	**Precio/Hora**	**Tarea**	**Horas de trabajo**	**Total**
Programador	1	10 €/hora	Resolver dudas funcionamiento	…	…
Programador	1	20 €/hora	Cambio funcionalidad aplicación	…	…

Así pues, la aplicación tendrá un coste inicial de 4950 euros, a lo que habría que añadir 300 euros anuales desde la fecha de implantación y la parte variable en función de las necesidades que les surjan con la utilización del programa.

Cabe mencionar que en realidad se han empleado más horas para la realización del proyecto, puesto que en lo anteriormente comentado no se ha incluido el documento explicativo del proyecto (duración aproximada de 90 horas).

Sin embargo el presupuesto anterior se ha elaborado con fines comerciales, y en ese ámbito el documento explicativo del proyecto de fin de grado no tiene cabida, siendo esa la razón por la que su realización no se ha incluido en el cómputo de horas.

7. CONCLUSIONES

La asignación de alumnos, ofertas y profesores antes de desarrollar la aplicación se realizaba manualmente y ocupaba durante una gran cantidad de tiempo a profesores en una tarea larga y monótona, que les quitaba tiempo para otros menesteres. La razón por la que se desarrolló la aplicación fue precisamente para sustituir esa tarea manual en una tarea que se realice de forma automática, en pocos minutos, y de forma mucho más eficiente.

Así pues, podemos decir que hemos cumplido íntegramente el objetivo inicial marcado para el desarrollo de la aplicación. A continuación explicamos en líneas generales las fases del proyecto.

En cuanto al desarrollo de la aplicación, decir que su comienzo fue muy dificultoso ya que yo no tenía conocimiento alguno del funcionamiento de los Practicum, Grados, asignaciones, etc. Es por ello por lo que se requirió mucho tiempo y trabajo en el diseño de una BASE DE DATOS que fuese eficiente y cumpliese con los requisitos establecidos, una tarea laboriosa y a veces incluso desesperante teniendo en cuenta que partíamos de cero.

Una vez hecho el diseño de la base de datos, el segundo paso fue la LECTURA DE FICHEROS. Aquí también hubo que invertir mucho tiempo puesto que los ficheros de entrada de datos son de diferentes formatos, y hubo que realizar un control de errores en todos y cada uno de ellos (5 tipos en total).

Sin embargo, aunque se dedicaron también muchas horas a esta parte del desarrollo, fue mucho más llevadera que la anterior debido a que esta tarea pudo dividirse en fases (una por tipo de fichero de entrada) mientras que la anterior era constantemente un “todo o nada”.

Con el diseño de la base de datos y la lectura de ficheros realizada, entramos a programar el primer gran bloque del proyecto, que es el diseño del Algoritmo ALUMNO-OFERTA. Como puede verse en la elaboración del presupuesto fue la fase en la que más tiempo se invirtió y hubo que realizar muchas llamadas y preguntas a lo largo de su desarrollo al tutor hasta que quedó claro el procedimiento. La mayor

dificultad de esta fase radicó en la posibilidad contemplada por el tutor de realizar asignaciones manualmente de alumnos y ofertas. Este modo de asignación lleva implícito la posibilidad de ejecutar el algoritmo en diferentes momentos, es decir, hubo que realizar una muy buen gestión de las ofertas que ya están asignadas y de las que aún están libres para los alumnos, complicando ello el diseño del algoritmo puesto que al realizar la asignación en diferentes pasos hay que almacenar en todo momento los cambios realizados.

El segundo gran bloque del desarrollo fue la programación del algoritmo PROFESOR-OFERTA. Al igual que la anterior fase hubo que realizar gran cantidad de consultas al tutor hasta aclarar la forma de asignar este proceso, por lo que también se invirtió gran cantidad de horas. Esta fase se hizo más amigable no por la dificultad (el procedimiento lleva una complejidad similar) sino porque en muchos aspectos como las interacciones con la base de datos era similar a la asignación alumno-oferta, por lo que una buena parte del código ya estaba programado.

También se quiso contemplar la posibilidad de ejecutar el algoritmo en diferentes momentos como en el anterior bloque, pero su dificultad fue reducida considerablemente puesto que ya partíamos de la experiencia del anterior algoritmo.

El último paso fue la GENERACIÓN DE RESULTADOS. Aquí la mayor parte del tiempo utilizado fue debido a que se generan un total de 9 ficheros resultantes cada uno con un formato específico. Esta tarea fue la más sencilla de todas debido a que ya partíamos de la experiencia adquirida anteriormente de interaccionar con archivos en formato CSV y Excel. Así pues, la razón de que se emplearan en esta fase 25 horas no fue debido a la dificultad sino a la monotonía de código y al volumen de trabajar con 9 ficheros y su creación en diferentes momentos del algoritmo.

Mencionar que para la realización del proyecto he utilizado gran cantidad de conocimientos adquiridos durante la carrera como el uso de la plataforma Netbenas, el lenguaje Java, base de datos MySQL, interacción entre Java y phpMyAdmin, desarrollo de algoritmos en Java, interfaces de Netbeans, importar ficheros Excel, exportar ficheros Excel, etc.

Para finalizar, decir que la aplicación consta de una interfaz amigable y

sencilla de cara un usuario básico y que al mismo tiempo cumple con todas las especificaciones requeridas. Así mismo, también se ha dotado a la aplicación de una gran escalabilidad para evitar problemas posteriores a su implantación, y de un control de errores exhaustivo durante todo el proceso.

En mi opinión se ha realizado un buen trabajo y con una sensación general muy positiva y gratificante. Esta sensación se debe no solo a la gran cantidad de tiempo que se va a ahorrar en futuros cursos académicos en asignaciones manuales, sino también a nivel personal puesto que me ha permitido aprender gran cantidad de aspectos importantes de cara al futuro como el manejo de gran cantidad de información en base de datos desde Java, modificación de tablas, inserciones, etc.

8. BIBLIOGRAFÍA

- Elmasri, R., Navathe, S.B., *Fundamentos de Sistemas de Bases de Datos*, 5ª edición, Pearson Education, 2008.

- Pons Capote et al. *Introducción a las bases de datos: el modelo relacional*. Thomson Paraninfo, 2005.

- Bruce Eckel., *Think in Java - Piensa en Java*, 4ª edición, Pearson Education, Prentice Hall, 2007.

- Quest. *Toad Data Modeler*, 2017. Accessed may 2017. https://www.quest.com/mx-es/products/toad-data-modeler/

- Download 3K. *Toad Data Modeler 5.2.4.27*, 2016. Accessed may 2017. http://www.download3k.es/Negocios-y-Finanzas/Bases-de-Datos/Download-Toad-Data-Modeler.html

- SCRIBD. R. Monago., *Diseño de base de datos relacionales utilizando herramientas gráficas.*, 2017. Accessed apr 2017. https://es.scribd.com/document/6495177/Toad-Data-Modeler

- Culturacion. *Qué es y para que sirve MySQL*, 2015. Accessed apr 2017. http://culturacion.com/que-es-y-para-que-sirve-mysql/

- PaCko. *Características de MYSQL*. 2017. Accessed may 2017. https://packo.wikispaces.com/Caracteristicas+de+MYSQL

- Brandominius. *[Manual] Sentencias básicas en MySQL*. 2015. Accessed may 2017. http://www.brandominus.com/manual-sentencias-basicas-en-mysql/

- Wikipedia. *phpMyAdmin.*2017. Accessed apr 2017. https://es.wikipedia.org/wiki/PhpMyAdmin

- Wikipedia. *Netbeans.* 2016. Accessed jun 2017. https://es.wikipedia.org/wiki/NetBeans

- Netbeans. *Información NetBeans IDE 6.1.*, 2017. Accessed may 2017. https://netbeans.org/community/releases/61/index_es.html

- Sebastian Gomez. *Creación y personalización de interfaces gráficas usando Netbeans.* 2015. Accessed apr 2017. http://www.sebastian-gomez.com/java/creacion-y-personalizacion-de-interfaces

ANEXO

MANUAL DE USUARIO

INDICE

1. INSTALACIÓN XAMPP

1.1. DESCARGA E INSTALACIÓN

El paquete XAMPP es un entorno gratuito que es necesario para la ejecución de la aplicación, puesto que contiene phpMyAdmin que es la herramienta de MySQL con la que el programa va a realizar las interacciones con la base de datos. Así pues, lo que haremos será ingresar en la siguiente página:

https://www.apachefriends.org/es/index.html

Una vez allí seleccionaremos la versión a descargar en función de nuestro sistema operativo. En nuestro caso el utilizado es Windows por lo que la versión a descargar será la seleccionada con un círculo amarillo.

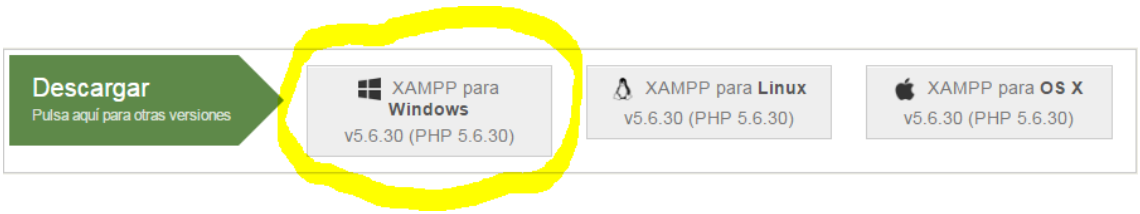


Fig. 1. Posibilidades de SSOO para la descarga

Al seleccionar en alguna de las versiones anteriores, se comenzará a descargar automáticamente un fichero con los archivos necesario para proceder a su instalación.

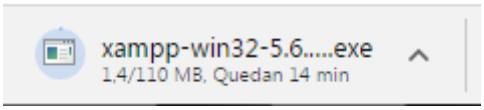


Fig. 2. Enlace de descarga en el explorador

Tras esperar unos minutos a que se realice la descarga, abriremos el archivo y nos encontraremos una ventana en la que se nos permite modificar parámetros de la instalación.

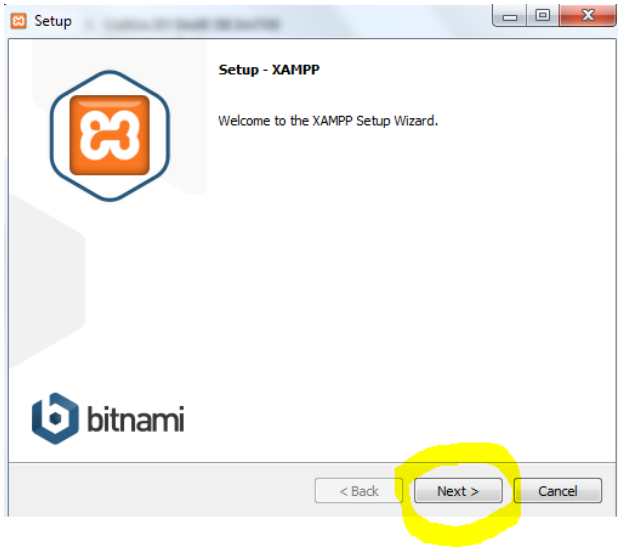


Fig. 3. Ventana de instalación

A continuación aparecerá una ventana con los diferentes entornos que podemos instalar dentro del paquete XAMPP. Por defecto aparecen seleccionados todos, pero para nuestro proyecto solo son necesarios los que están en color amarillo, por lo que serán los únicos que instalemos con el fin de reducir el tiempo de instalación.

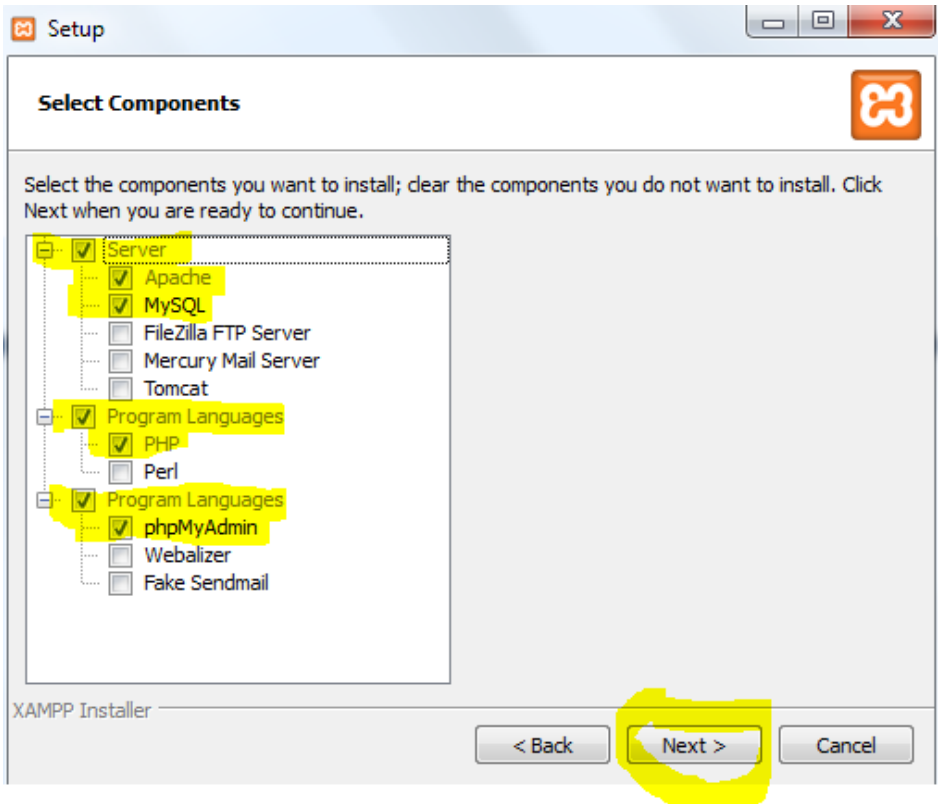


Fig. 4. Posibles paquetes a instalar

En el siguiente paso se debe especificar la ruta donde será realizada la instalación, que en nuestro caso dejaremos la que está por defecto.

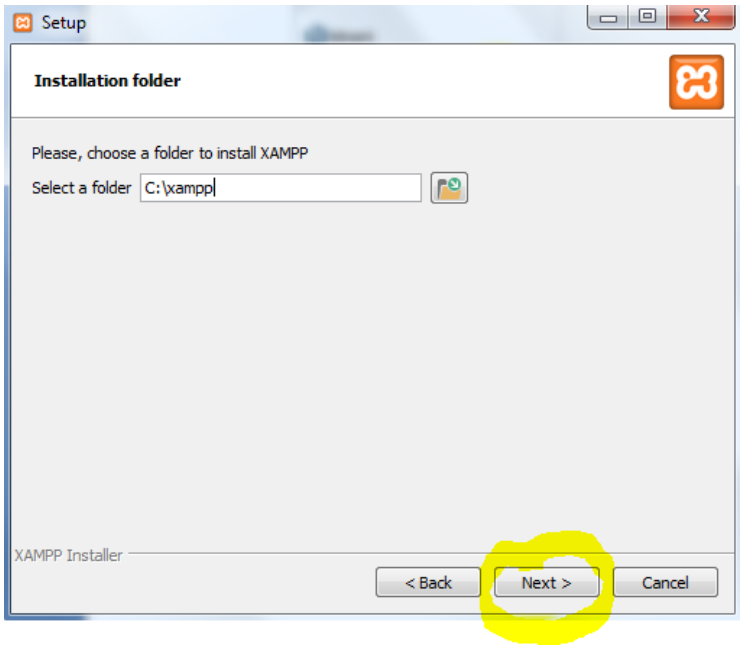


Fig. 5. Ruta de destino del programa a instalar

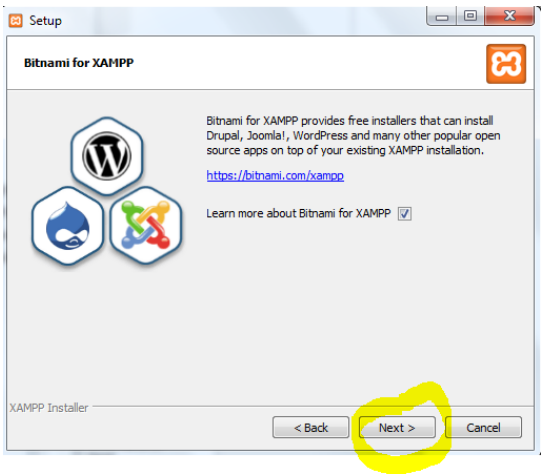


Fig. 6. Ventana de instalación

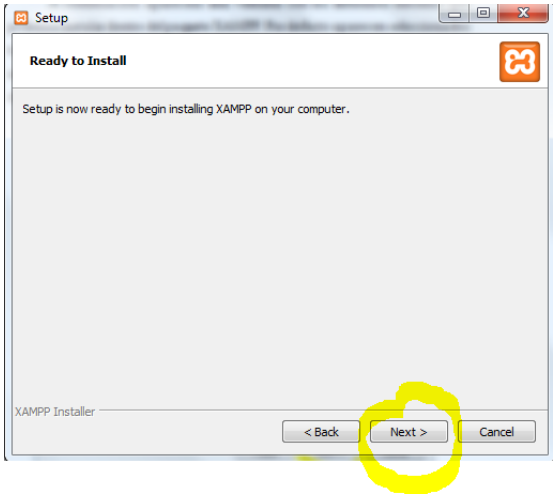


Fig. 7. Ventana de instalación

Tras realizar lo indicado en las 3 imágenes anteriores, comenzará la instalación del paquete.

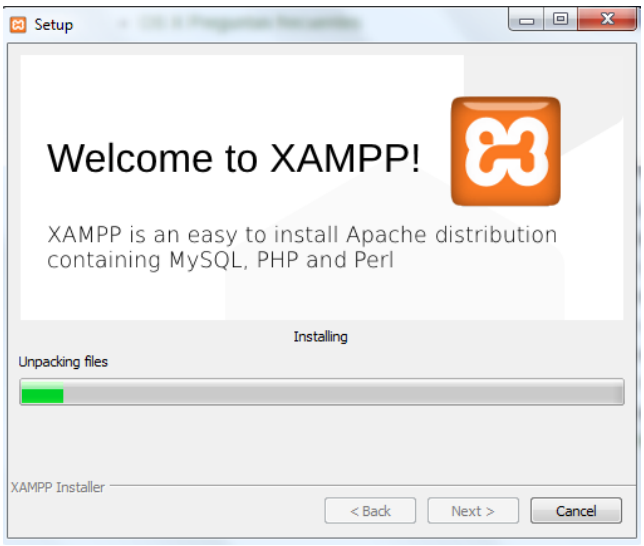


Fig. 8. Ventana del progreso de instalación

Tras esperar unos minutos a que se haya instalado el paquete en nuestro ordenador, abriremos la aplicación que se nos ha descargado y nos encontraremos la ventana mostrada en la siguiente imagen. En ella seleccionaremos los dos botones “Start” que están sombreados en amarillo.

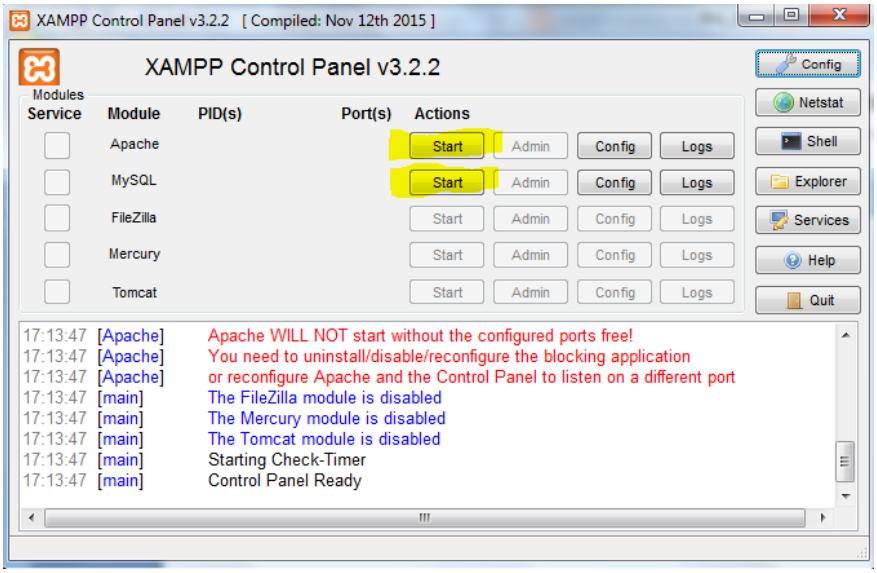


Fig. 9. Panel central de la aplicación XAMPP

Una vez hecho eso, seleccionaremos el botón “Admin” que se encuentra sombreado en amarillo en la fila de “MySQL” tal y como se muestra en esta imagen:

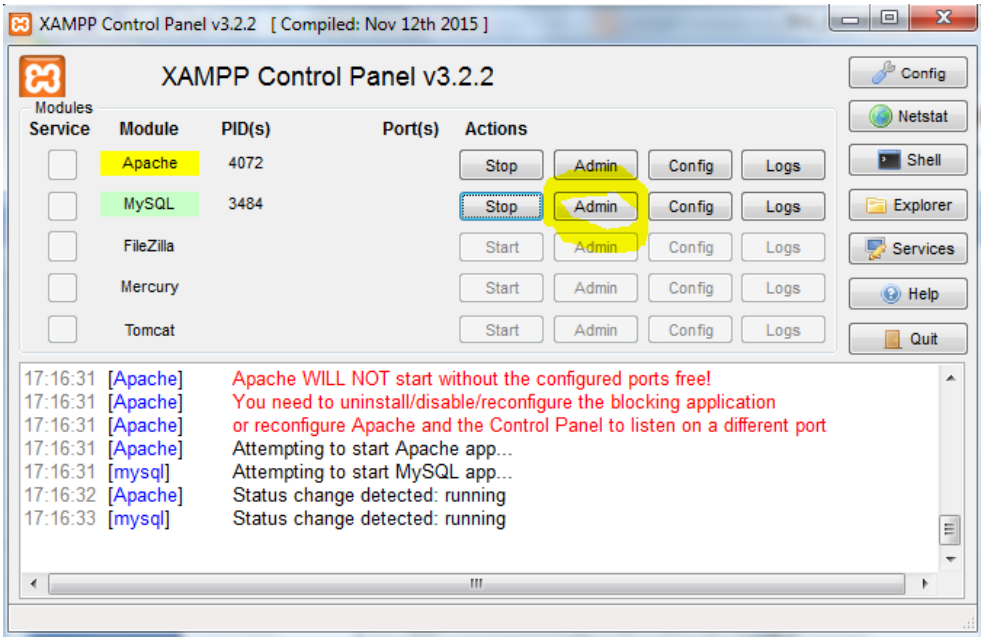


Fig. 10. Panel central de la aplicación XAMPP

1.2. CONFIGURACIÓN

Una vez hecho esto, se nos abrirá en el navegador que tengamos por defecto phpMyAdmin, que será desde donde controlemos la base de datos con la que va a interactuar el programa durante su ejecución.

Fig. 11. Panel central de phpMyAdmin en el explorador

Como puede verse en la anterior imagen, aparecen a la izquierda una lista de las bases de datos que hay en ese momento. En nuestro caso las bases de datos serían "2" y "3". A continuación procederemos a crear una nueva base de datos.

Ahora, dentro del entorno phpMyAdmin, procederemos a la creación de la base de datos y será allí donde importemos el archivo proporcionado en el proyecto con toda la información acerca de las tablas y de las relaciones

Fig. 12. Especificación de nombre de base de datos

Seleccionaremos un nombre de la nueva base de datos, y pulsaremos en Crear. En lugar de hacerlo en modo interfaz, también podríamos crear la base de datos mediante sentencia, escribiendo en la consola SQL el siguiente comando:

CREATE DATABASE nuevaBase

Ahora, una vez creada la base de datos, sería recomendable asignarle una contraseña puesto que por defecto no viene ninguna, aunque para el desarrollo del proyecto no se ha asignado ninguna. Ahora ya tenemos nuestra nueva base de datos creada, y pasaremos a lo siguiente, que es crear todas las tablas.

IMPORTANTE

El nombre de la base de datos no puede ser cualquiera. En el programa se ha establecido que la base de datos se llame "**nuevaBase**" (como en el ejemplo anterior), así que ese debe de ser su nombre obligatoriamente en caso de que no se quiera modificar el código del programa.

Para crear todas las tablas y relaciones con las que la aplicación debe trabajar a continuación, lo que haremos será IMPORTAR el archivo SQL proporcionado en la carpeta del trabajo de fin de grado. El archivo se llama **"estructuraBBDD.sql**".

Fig. 13. Selección del archivo SQL a importar

Acto seguido, abajo pulsaremos el botón continuar, y ello implicará que se habrán ejecutado todas las sentencias de creación de tablas, por lo que habrá finalizado la creación de la estructura de nuestra base de datos, tal que así:

Fig. 14. Mensajes de resultado de la inserción del archivo SQL

En este momento ya tenemos todas las tablas y relaciones de la base de datos con la que la aplicación debe interactuar durante su ejecución, por lo que ya podremos pasar a correr el programa. A partir de ahora ya NO ES NECESARIO que el usuario realice ningún tipo de interacción con la base de datos en el navegador, solo tiene que dejar la pestaña abierta para que la aplicación pueda interactuar con ella.

2. EJECUCIÓN DE LA APLICACIÓN

Para iniciar la aplicación habrá que ir a la carpeta **Aplicación_Ejecutable**, y dentro de ella seleccionar el archivo **ejecutable.exe**

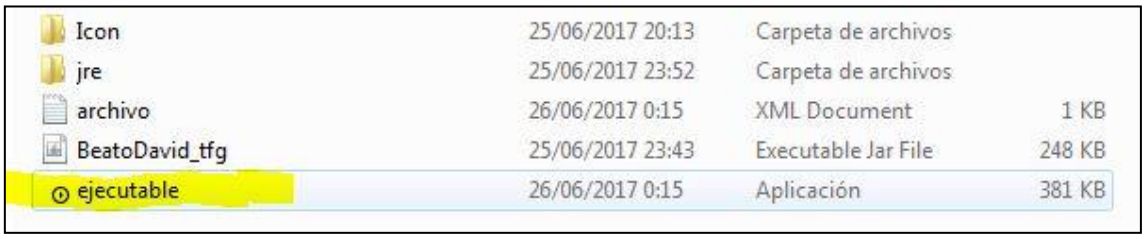


Fig. 15. Archivos de la carpeta Aplicación_Ejecutable

En caso de que el antivirus proceda a analizar el archivo y tarde demasiado tiempo, cancelar dicho análisis. Al iniciar la aplicación se le mostrará al usuario el siguiente panel central desde donde tendrá acceso a todas las funcionalidades.

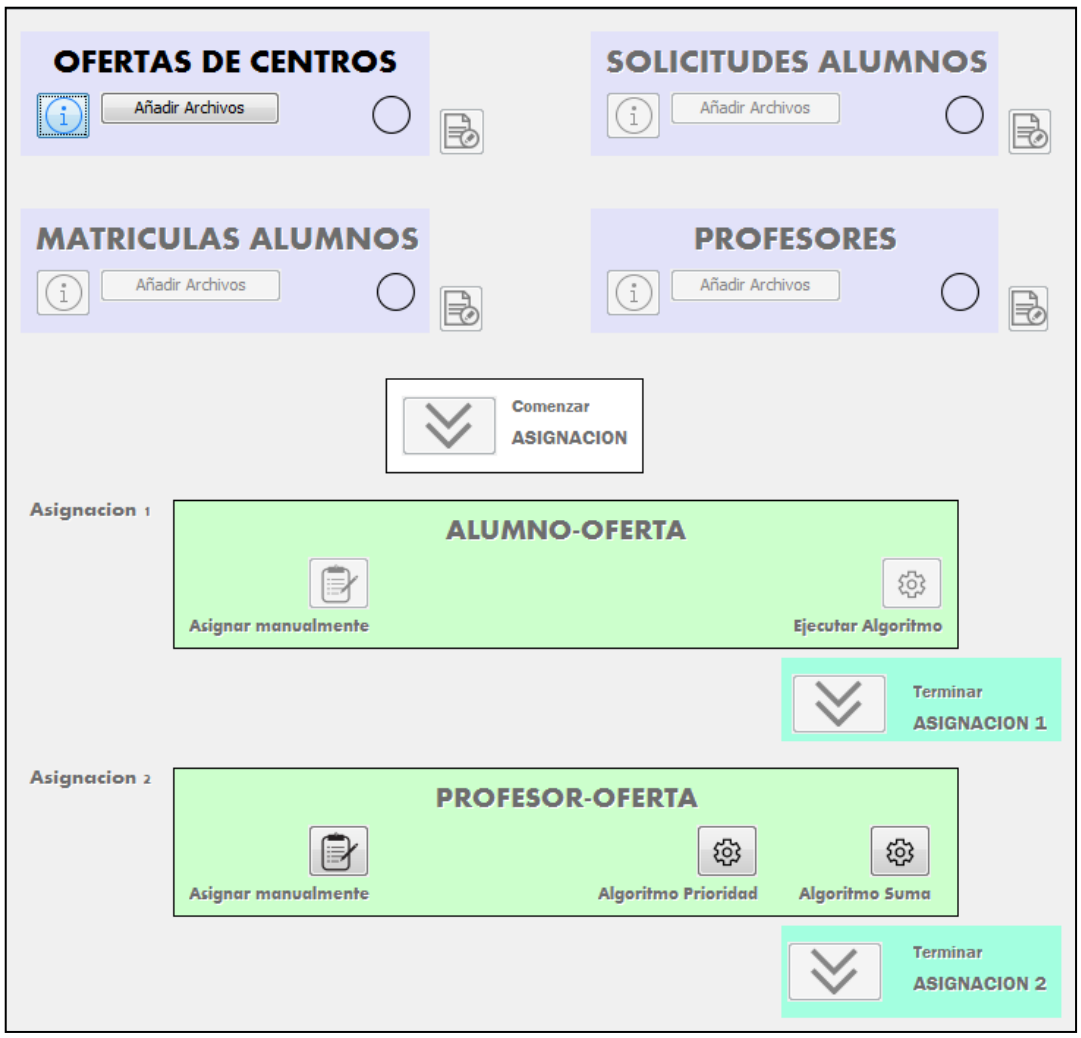


Fig. 16. Panel central de la aplicación desarrollada

3. ELECCION DE FICHEROS

El primer paso a realizar es la elección de los ficheros de entrada donde se deben especificar las direcciones de los datos de entrada con los que va a trabajar la aplicación. Los 4 tipos de ficheros a insertar serán centros, formularios, alumnos y profesores. La elección de todos ellos se realiza en esta sección:

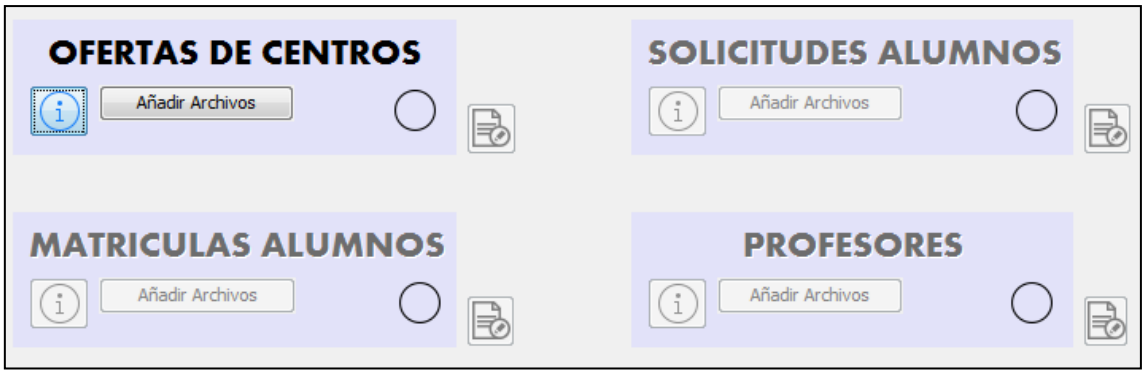


Fig. 17. Sección de la interfaz dedicada al input de ficheros

Dentro de esta sección, los siguientes botones sirven para realizar las siguientes funcionalidades:

Anula los archivos especificados en la sección donde se pulse, de forma que permite al usuario volver a especificar dichos ficheros.

Al pasar el cursor por encima aparece información acerca de los tipos de archivos que deben especificarse en dicha sección.

3.1. CENTROS

Fig. 18. Selección de archivo con datos de centros

En esta sección seleccionaremos los archivos con los centros y las ofertas de Guadalajara y de la Comunidad de Madrid. Para proceder a realizar la inserción, pulsamos el botón "**Añadir archivos**".

Una vez dentro, como puede verse en la siguiente imagen, simplemente seleccionaremos el archivo que contiene los centros de Guadalajara, y el archivo que contiene los centros de la Comunidad de Madrid.

Fig. 19. Selección de la ruta archivo con datos de centros

En caso de que solo indiquemos un único archivo en lugar de los 2 necesarios, al pulsar aceptar no nos dejará abandonar la ventana. Es decir, es obligatorio que se indiquen dos ficheros, y que además estos sean diferentes.

Una vez especificada la ruta correctamente, pulsaremos aceptar y volveremos al panel central. En caso de que la especificación de archivos haya sido correcta, aparecerá de la siguiente forma:

Fig. 20. Resultado después de la selección de archivos con datos de centros

Llegados a este punto, podemos pasar a la siguiente fase que es la correspondiente a la elección de formularios.

Nota: La única comprobación que se hace en esta sección al especificar los ficheros es que se hayan insertado dos, y que además estos sean diferentes. Esto quiere decir que en caso de que el fichero no tuviese el formato correcto, daría error más adelante.

3.2. FORMULARIOS

Fig. 21. Selección de archivo con datos de elecciones de alumnos

En esta sección seleccionaremos los archivos con los formularios de los alumnos, es decir, su lista de elecciones ordenadas por orden de preferencia. Para proceder a realizar la inserción, pulsamos el botón "**Añadir archivos**".

Fig. 22. Selección de rutas de archivos con datos de elecciones alumnos

Una vez dentro, iremos añadiendo archivos (los que sean necesarios) con las elecciones de los alumnos a la lista de archivos que aparece, de esta forma:

 Añadir archivo a la lista

 Eliminar todos los archivos de la lista

Una vez que se hayan especificado en la lista todos los archivos que van a utilizarse como entrada, pulsaremos aceptar y volveremos al panel central. En caso de que la especificación haya sido correcta, aparecerá de la siguiente forma:

Fig. 23. Resultado después de la selección de rutas de archivos con datos de elecciones de alumnos

Llegados a este punto, podemos pasar a la siguiente fase que es la correspondiente a la elección de matrículas.

Nota: La única comprobación que se hace en esta sección al especificar los ficheros es que no haya ninguno repetido. Esto quiere decir que en caso de que alguno de los ficheros no tuviese el formato correcto, daría error más adelante.

3.3. ALUMNOS

Fig. 24. Selección de rutas de archivos con alumnos matriculados

En esta sección seleccionaremos los archivos con las matrículas de los alumnos, es decir, la lista de alumnos matriculados en un curso académico y donde se indica el grado matriculado, el practicum, la nota media, etc. Para proceder a realizar la inserción, pulsamos el botón "**Añadir archivos**".

Fig. 25. Panel de selección de rutas de archivos con alumnos matriculados

Una vez dentro, iremos añadiendo archivos (los que sean necesarios) con los alumnos matriculados a la lista de archivos que aparece. En el presente proyecto los archivos a especificar son 17 (uno por cada grado y practicum diferente). La adicción de archivos se realizará de esta forma:

 Añadir archivo a la lista

 Eliminar todos los archivos de la lista

Una vez que se hayan especificado en la lista todos los archivos que van a utilizarse como entrada, pulsaremos aceptar y volveremos al panel central. En caso de que la especificación haya sido correcta, aparecerá de la siguiente forma:

Fig. 26. Resultado después de la selección de rutas de archivos con alumnos matriculados

Llegados a este punto, podemos pasar a la siguiente fase que es la correspondiente a la elección de los profesores.

Nota: La única comprobación que se hace en esta sección al especificar los ficheros es que no haya ninguno repetido. Esto quiere decir que en caso de que alguno de los ficheros no tuviese el formato correcto, daría error más adelante.

3.4. PROFESORES

Fig. 27. Selección de rutas de archivos con profesores y sus preferencias

En esta sección seleccionaremos los archivos con la información de los profesores, es decir, los profesores que van a tutorizar los practicum (indicando el número a tutorizar, su especialidad, etc.).

Para proceder a realizar la inserción, pulsamos el botón "**Añadir archivos**".

Fig. 28. Panel de selección de rutas de archivos con profesores y sus preferencias

Una vez dentro, iremos añadiendo archivos (en nuestro proyecto 1 pero pueden ser más) con los profesores ofertados a tutorizar practicums a la lista de archivos que aparece. La adicción de archivos se realizará de esta forma:

 Añadir archivo a la lista

 Eliminar todos los archivos de la lista

Una vez que se hayan especificado en la lista todos los archivos que van a utilizarse como entrada, pulsaremos aceptar y volveremos al panel central. En caso de que la especificación haya sido correcta, aparecerá de la siguiente forma:

Fig. 29. Resultado de selección de rutas de archivos con profesores y sus preferencias

Llegados a este punto, ya habremos terminado la especificación de los 4 tipos de ficheros de entrada que necesita la aplicación, por lo que la siguiente fase es la correspondiente a la carga en la base de datos de todos los ficheros que hemos especificado en eso 4 pasos anteriores.

Nota: La única comprobación que se hace en esta sección al especificar los ficheros es que no haya ninguno repetido. Esto quiere decir que en caso de que alguno de los ficheros no tuviese el formato correcto, daría error más adelante.

4. CARGA EN BASE DE DATOS

Llegados a este punto, lo único que hemos hecho es especificar la RUTA de los archivos de donde se va a extraer la información de entrada, pero aún no la hemos extraído. Esa extracción y carga en la base de datos es lo que se realiza en esta fase. Así pues, para realizar la carga masiva de información a la base de datos, seleccionamos el siguiente botón del panel de control:

Fig. 30. Botón para comenzar el proceso de asignación

Una vez pulsado, se nos abrirá una nueva ventana que nos permitirá especificar dos valores antes de dar comienzo a la carga de datos.

Fig. 31. Panel de especificación de datos previos a comenzar la asignación

- LETRA DE PREFERENCIA: Letra a partir de la cual serán ordenados los profesores en relación a la primera de su apellido.

- DIRECTORIO DE RESULTADOS: Carpeta donde serán almacenados los ficheros que se generen durante la ejecución del programa. Se recomienda crear una carpeta nueva para localizar los ficheros más rápido.

Una vez especificados estos dos valores, pulsaremos "ACEPTAR" y se nos abrirá un diálogo como el siguiente:

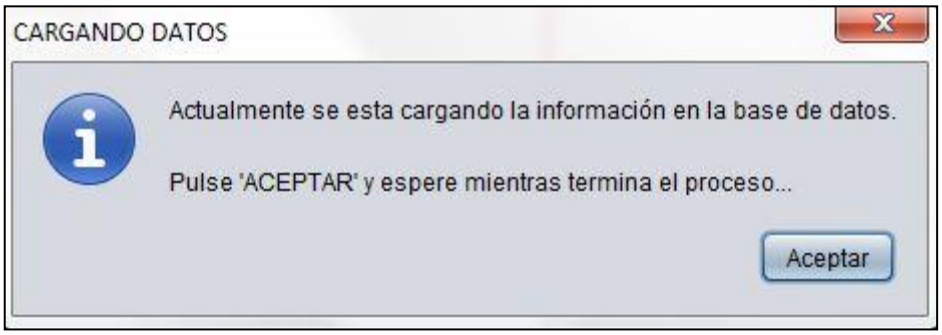


Fig. 32. Mensaje informativo al usuario. Cargando datos en la aplicación.

IMPORTANTE

El usuario **DEBE PULSAR ACEPTAR** para dar comienzo a la carga de información en la base de datos. Hasta que el usuario no pulse "Aceptar" no se realizará ninguna acción.

Una vez que el usuario pulse "Aceptar" lo único que tiene que hacer es esperar a que termine el proceso. Por la apariencia de la aplicación parece que al pulsar "Aceptar" la aplicación no ha realizado absolutamente nada, sin embargo, internamente está realizando todas las inserciones en la base de datos que son ocultas de cara al usuario.

Este proceso puede tardar **VARIOS MINUTOS** debido a la gran cantidad de datos especificados en pasos anteriores. Después de esperar durante un tiempo, aparecerá alguno de estos mensajes:

MENSAJE 1

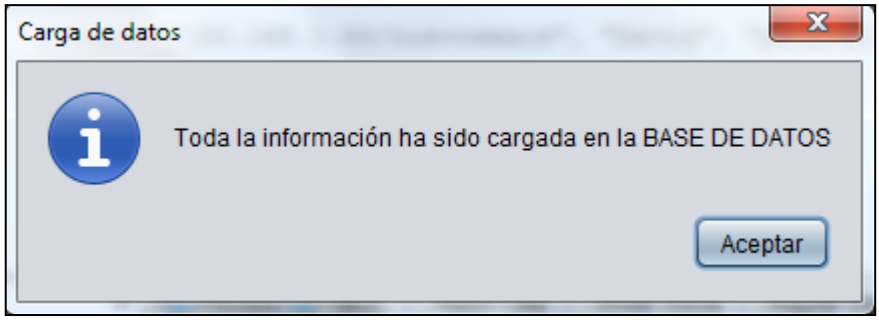


Fig. 33. Mensaje informativo al usuario. Información cargada en la aplicación.

Este mensaje nos indica que esa lista de ficheros ha sido creada en la carpeta especificada como DIRECTORIO DE RESULTADOS. (Consultar ficheros generados en el apartado "*7. FICHEROS RESULTANTES*"). En este momento el programa ya tiene todos los datos con la que va a trabajar, pero AÚN NO HA COMENZADO LA ASIGNACIÓN, eso se hará en la siguiente fase. Es decir, hasta ahora lo único que ha hecho ha sido almacenar la información con la que trabajar. El usuario debe pulsar "Aceptar", y volverá al panel central.

MENSAJE 2

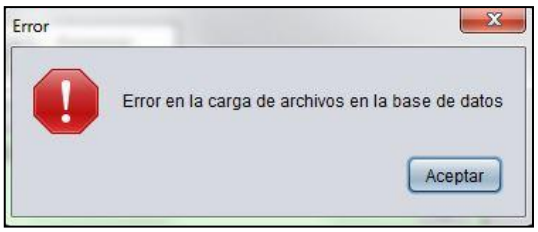


Fig. 34. Mensaje informativo al usuario. Error en la carga de datos.

Este mensaje nos indica que ha habido algún error en la carga de datos. Este mensaje puede deberse a que phpMyAdmin se ha saturado por alguna razón, o bien porque alguno de los ficheros introducidos no tiene el formato correcto.

Se recomienda al usuario que cierre este diálogo y vuelva a realizar la carga de datos (sin cambiar ningún fichero). Si tras este segundo intento, vuelve a dar el mismo error, quiere decir que los alguno de los ficheros introducidos no tiene el formato correcto y por tanto el usuario debe volver a especificar la lista de archivos.

5. Asignación ALUMNO-OFERTA

Una vez terminada satisfactoriamente la carga de datos, procederemos a realizar el primer bloque de la asignación, que es la asignación ALUMNO-OFERTA. La asignación entre alumnos y ofertas se realiza desde la siguiente sección del panel central:

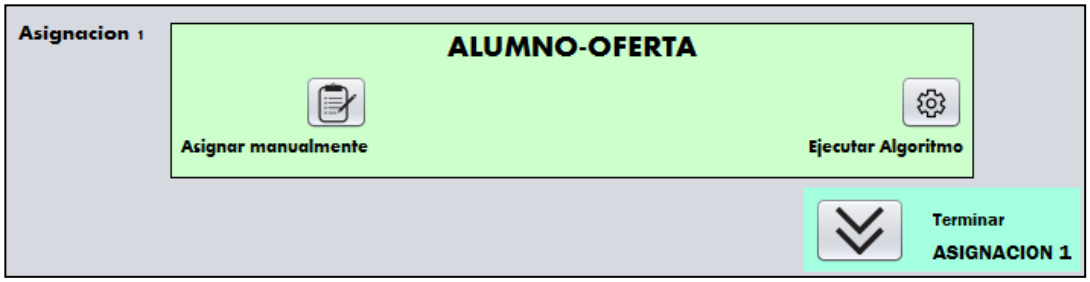


Fig. 35. Sección del panel central dedicada a la asignación entre alumnos y ofertas.

Como puede verse, la asignación puede realizarse de dos formas, que dan lugar a los siguientes dos apartados. El orden de realizar cada una de ellas es indiferente, es decir, el usuario puede ejecutar solamente asignación manual, solamente asignación automática, ejecutar primero manual y luego automática, ejecutar automática y después manual, etc. tantas veces como se quiera.

5.1. Asignación AUTOMÁTICA

Fig. 36. Botón para la ejecución automática del algoritmo de asignación entre alumno y oferta.

Una vez que el usuario pulse el botón se comenzar aparecerá el siguiente diálogo:

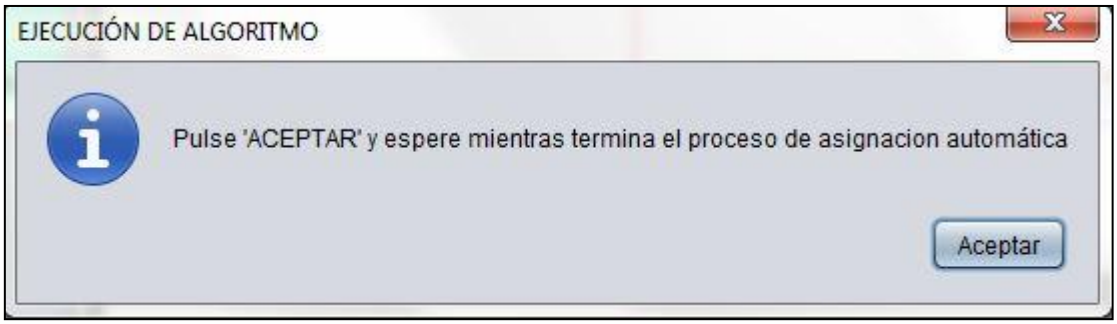


Fig. 37. Mensaje informativo al usuario. El proceso de asignación automática se está ejecutando.

IMPORTANTE

El usuario **DEBE PULSAR ACEPTAR** para dar comienzo a la ejecución del algoritmo automático. Hasta que el usuario no pulse "Aceptar" no se realizará ninguna acción.

Una vez que el usuario pulse "Aceptar" lo único que tiene que hacer es esperar a que termine el proceso. Por la apariencia de la aplicación parece que al pulsar "Aceptar" la aplicación no ha realizado absolutamente nada, sin embargo, internamente está realizando todas las asignaciones entre alumno y oferta, solo que dicho proceso permanece oculto al usuario.

Este proceso puede tardar **VARIOS MINUTOS** debido a la gran cantidad de datos con los que trabajar. Al finalizar se nos mostrará el siguiente mensaje indicándonos los archivos que han sido modificados.

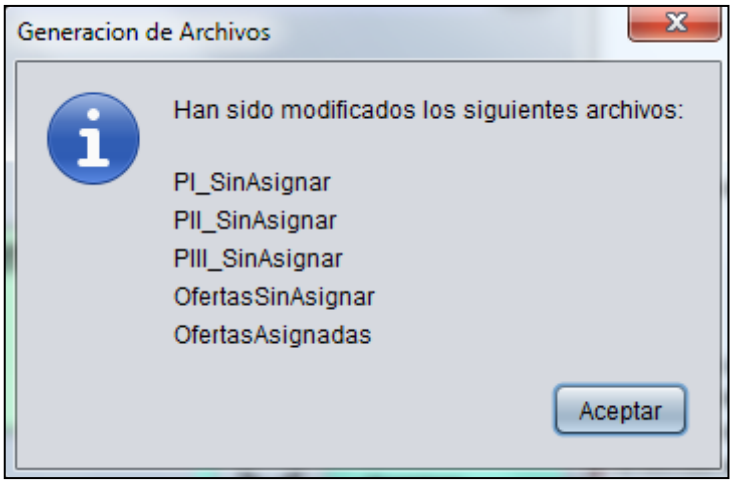


Fig. 38. Mensaje informativo al usuario. Lista de archivos que han sido modificados.

Puede consultarse el contenido de cada fichero resultante en el apartado "*7. FICHEROS RESULTANTES*".

5.2. Asignación MANUAL

Fig. 39. Botón para asignación manual entre alumno y oferta.

En este modo de asignación es el usuario quien establece manualmente las asignaciones entre alumnos y ofertas. Al pulsar el botón deberemos especificar el archivo en formato CSV donde se encuentren las asignaciones manuales que hemos hecho. El siguiente ejemplo ilustra cómo debe de ser el formato del fichero.

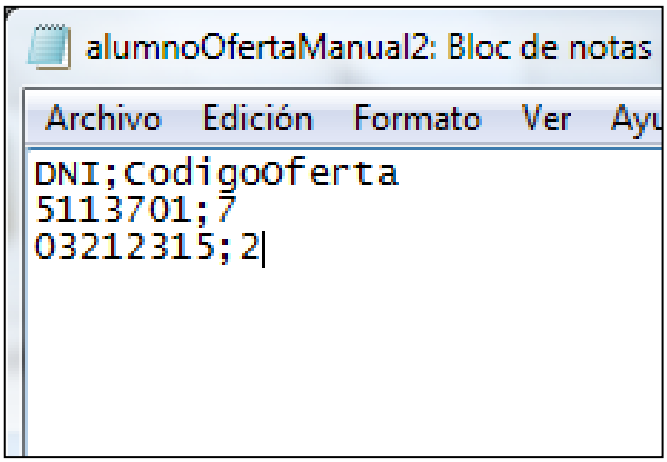


Fig. 40. Ejemplo de formato de archivo de entrada para asignación manual entre alumnos y ofertas.

En este ejemplo, el alumno con DNI=’5113701’ será asignado a la oferta con CódigoOferta=’7’, y el alumno con DNI=’03212315’ será asignado a la oferta con CódigoOferta=’2’.

Los DNI a introducir deben de ser alguno de los que aparecen en el fichero P**I_AlumnosSinAsignar.xlsx**, P**II_AlumnosSinAsignar.xlsx,** o bien en el fichero P**III_AlumnosSinAsignar.xlsx.**

Los códigos de oferta a especificar deben de ser alguno de los que aparecen en el fichero O**fertasSinAsignar.xlsx.**

Una vez que el usuario ha especificado la ruta del fichero CSV para la asignación manual, aparecerá el siguiente diálogo:

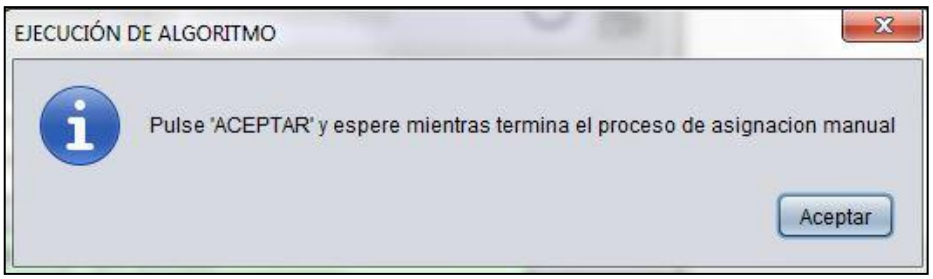


Fig. 41. Mensaje informativo. La asignación manual se está llevando a cabo.

IMPORTANTE

El usuario **DEBE PULSAR ACEPTAR** para dar comienzo a la ejecución del algoritmo manual. Hasta que el usuario no pulse “Aceptar” no se realizará ninguna acción.

Una vez que el usuario pulse “Aceptar” lo único que tiene que hacer es esperar a que termine el proceso. Por la apariencia de la aplicación parece que al pulsar “Aceptar” la aplicación no ha realizado absolutamente nada, sin embargo, internamente está realizando todas las asignaciones entre alumno y oferta que se han indicado manualmente, solo que dicho proceso permanece oculto al usuario.

Al finalizar se nos mostrará el siguiente mensaje indicándonos los archivos que han sido modificados, cuyo contenido puede consultarse en el apartado "*7. FICHEROS RESULTANTES*".

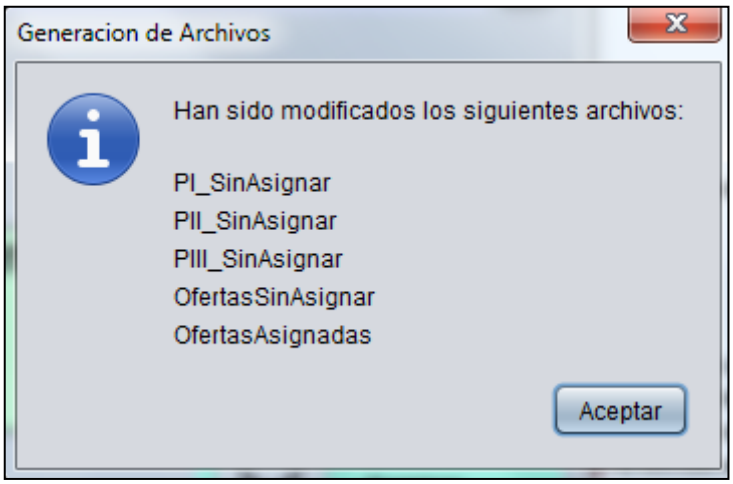


Fig. 42. Mensaje informativo. Lista de archivos que han sido modificados.

6. Asignación PROFESOR-OFERTA

Una vez terminada satisfactoriamente el primer bloque de la asignación, procederemos a realizar el SEGUNDO BLOQUE. La asignación entre profesores y ofertas se realiza desde la siguiente sección del panel central:

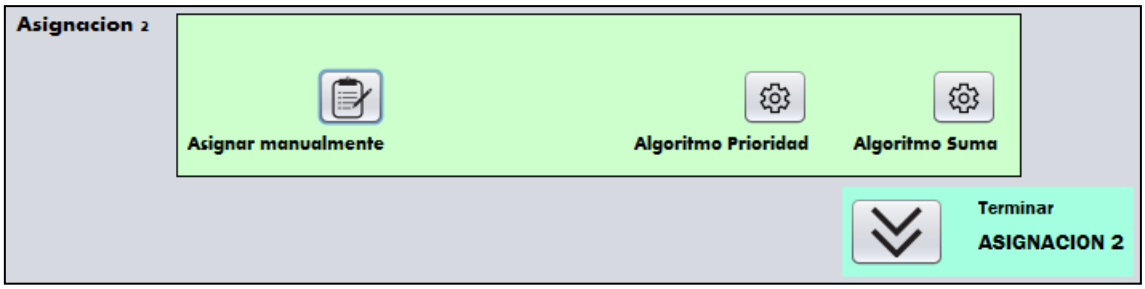


Fig. 43. Sección del panel dedicada a la asignación entre ofertas y profesores.

Como puede verse, la asignación puede realizarse de tres formas diferentes, que dan lugar a los siguientes tres apartados. El orden de realizar cada una de ellas es indiferente, es decir, el usuario puede ejecutar solamente asignación manual, solamente asignación automática, ejecutar primero manual y luego automática, ejecutar automática y después manual, etc. tantas veces como se quiera.

6.1. Asignación AUTOMÁTICA 1: Algoritmo SUMA

Fig. 44. Botón de asignación automática de profesores y ofertas por medio del algoritmo de suma.

Una vez que el usuario pulse el botón se comenzar aparecerá el siguiente diálogo:

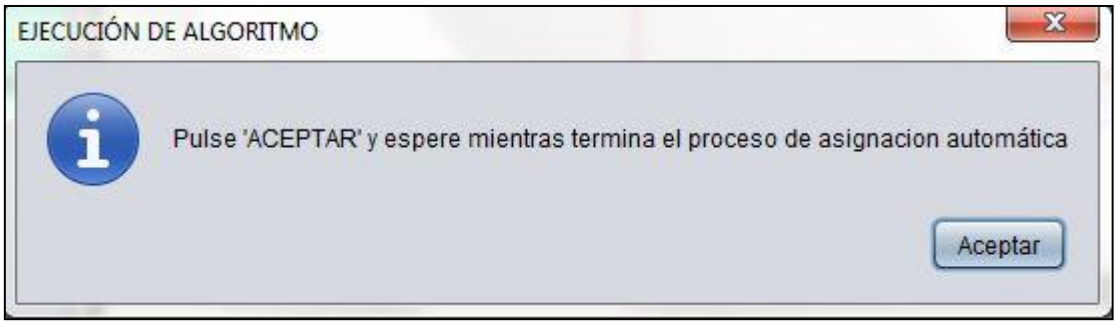


Fig. 45. Mensaje informativo. El algoritmo automático de suma se está ejecutando.

IMPORTANTE

El usuario **DEBE PULSAR ACEPTAR** para dar comienzo a la ejecución del algoritmo automático. Hasta que el usuario no pulse "Aceptar" no se realizará ninguna acción.

Una vez que el usuario pulse "Aceptar" lo único que tiene que hacer es esperar a que termine el proceso. Por la apariencia de la aplicación parece que al pulsar "Aceptar" la aplicación no ha realizado absolutamente nada, sin embargo, internamente está realizando todas las asignaciones entre profesor y oferta, solo que dicho proceso permanece oculto al usuario. Este proceso puede tardar **VARIOS MINUTOS** debido a la gran cantidad de datos con los que trabajar.

En caso de que aparezca un mensaje de advertencia como el siguiente quiere decir que el algoritmo ha realizado la asignación automática hasta llegar al profesor con DNI='2', es decir, este profesor no ha podido tutorizar en las zonas que deseaba los practicums que quería.

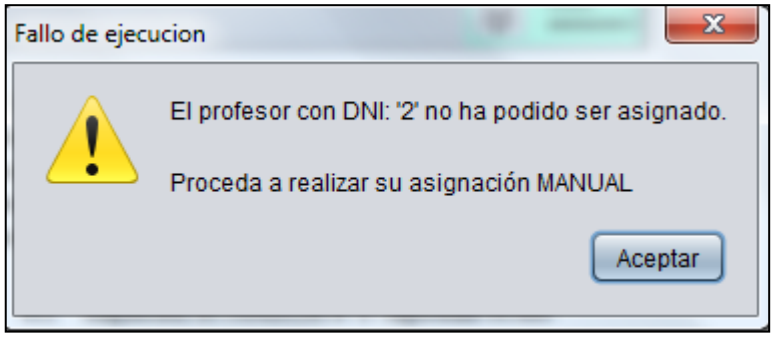


Fig. 46. Mensaje de advertencia. Un profesor en concreto no ha podido ser asignado automáticamente.

Una vez que aparezca este mensaje, el algoritmo se habrá detenido hasta el profesor en cuestión, y todos los profesores hasta él habrán sido asignados y exportados en el fichero de resultados. Ahora el usuario debe proceder a asignar manualmente este único profesor tal como se muestra en el apartado "*6.3. Asignación MANUAL*". Una vez realizada esa asignación, el usuario deberá volver a pulsar el botón de asignación automática para proseguir con las asignaciones donde se detuvieron.

Al finalizar se nos mostrará el siguiente mensaje indicándonos los archivos que han sido modificados.

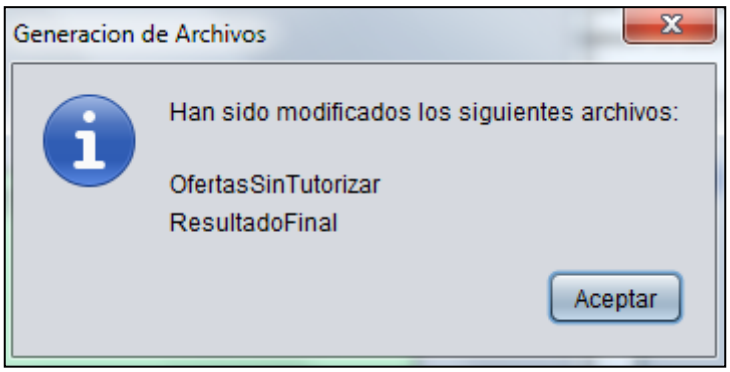


Fig. 47. Mensaje informativo. Lista de archivos que han sido modificados.

Puede consultarse el contenido de cada fichero resultante en el apartado "*7. FICHEROS RESULTANTES*".

6.2. Asignación AUTOMÁTICA 2: Algoritmo PRIORIDAD

Fig. 48. Botón de asignación automática entre profesores y ofertas por medio del algoritmo prioridad.

Una vez que el usuario pulse el botón se comenzar aparecerá el siguiente diálogo:

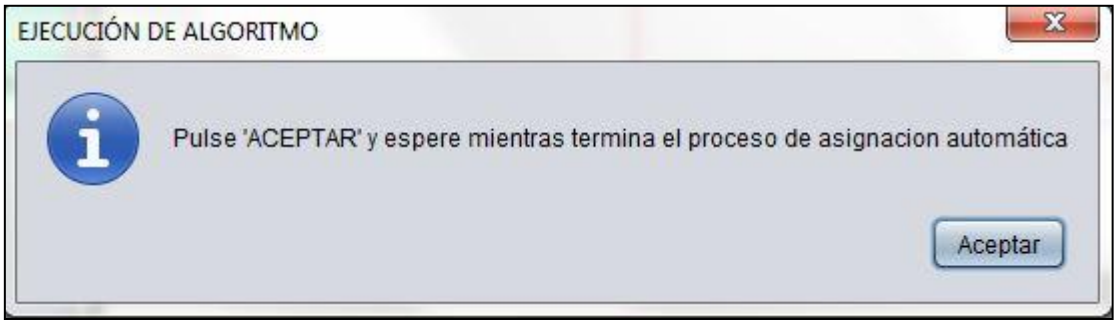


Fig. 49. Mensaje informativo. La asignación automática se está llevando a cabo.

IMPORTANTE

El usuario **DEBE PULSAR ACEPTAR** para dar comienzo a la ejecución del algoritmo automático. Hasta que el usuario no pulse "Aceptar" no se realizará ninguna acción.

Una vez que el usuario pulse "Aceptar" lo único que tiene que hacer es esperar a que termine el proceso. Por la apariencia de la aplicación parece que al pulsar "Aceptar" la aplicación no ha realizado absolutamente nada, sin embargo, internamente está realizando todas las asignaciones entre profesor y oferta, solo que dicho proceso permanece oculto al usuario. Este proceso puede tardar **VARIOS MINUTOS** debido a la gran cantidad de datos con los que trabajar.

En caso de que aparezca un mensaje de advertencia como el siguiente quiere decir que el algoritmo ha realizado la asignación automática hasta llegar al profesor con DNI='2', es decir, este profesor no ha podido tutorizar en las zonas que deseaba los practicums que quería.

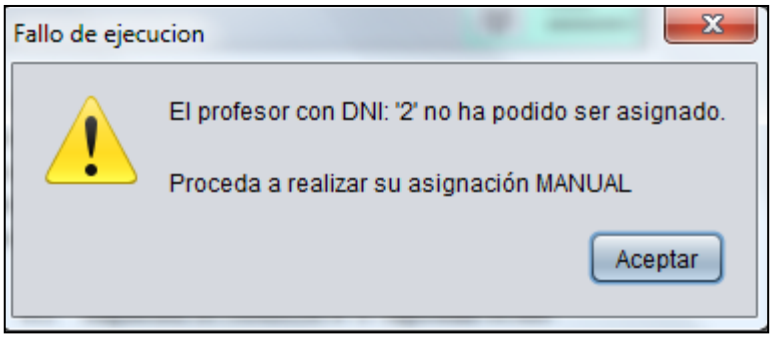


Fig. 50. Mensaje de advertencia. Un profesor en concreto no ha podido ser asignado automáticamente.

Una vez que aparezca este mensaje, el algoritmo se habrá detenido hasta el profesor en cuestión, y todos los profesores hasta él habrán sido asignados y exportados en el fichero de resultados. Ahora el usuario debe proceder a asignar manualmente este único profesor tal como se muestra en el apartado "*6.3. Asignación MANUAL*". Una vez realizada esa asignación, el usuario deberá volver a pulsar el botón de asignación automática para proseguir con las asignaciones donde se detuvieron.

Al finalizar se nos mostrará el siguiente mensaje indicándonos los archivos que han sido modificados.

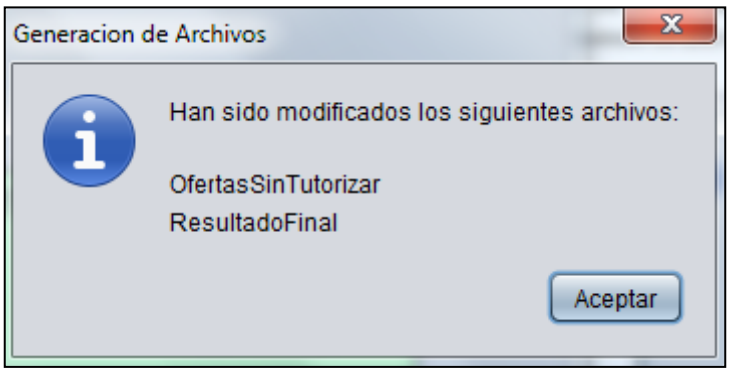


Fig. 51. Mensaje informativo. Lista de archivos modificados.

Puede consultarse el contenido de cada fichero resultante en el apartado "*7. FICHEROS RESULTANTES*".

6.3. Asignación MANUAL.

Fig. 52. Botón de asignación manual entre profesores y ofertas.

En este modo de asignación es el usuario quien establece manualmente las asignaciones entre profesores y ofertas. Al pulsar el botón deberemos especificar el archivo en formato CSV donde se encuentren las asignaciones manuales que hemos hecho. El siguiente ejemplo ilustra cómo debe de ser el formato del fichero.

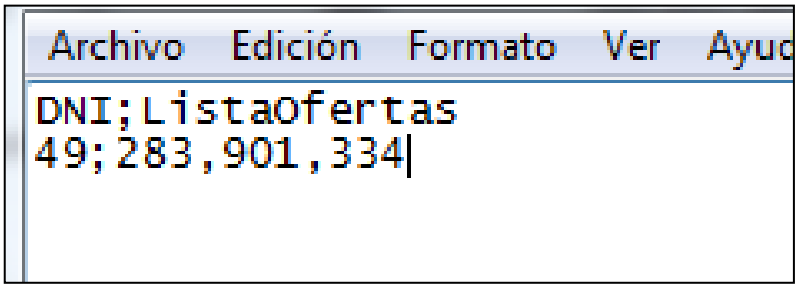


Fig. 53. Formato del fichero de entrada para asignación manual entre profesores y ofertas.

En este ejemplo, el profesor con DNI=’49’ será asignado a las ofertas con CódigoOferta=’283’, CódigoOferta=’901’ y CódigoOferta=’334’.

Los DNI a introducir deben de ser alguno de los que no aparezcan en el fichero **ResultadoFinal.xlsx** puesto que esos se corresponden con profesores ya asignados.

Los códigos de oferta a especificar deben de ser alguno de los que aparecen en el fichero O**fertasAsignadas.xlsx.**

Una vez que el usuario ha especificado la ruta del fichero CSV para la asignación manual, aparecerá el siguiente diálogo:

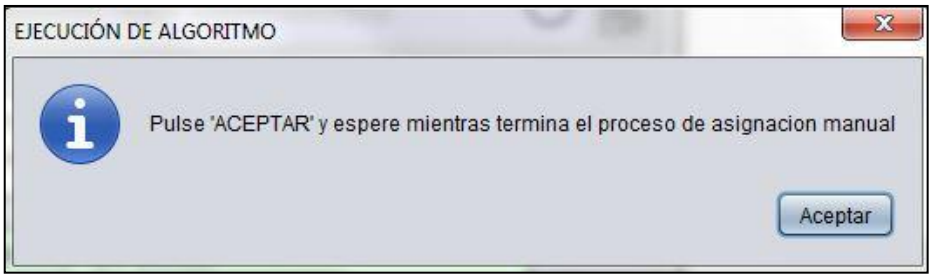


Fig. 54. Mensaje informativo. La asignación manual se está llevando a cabo.

IMPORTANTE

El usuario **DEBE PULSAR ACEPTAR** para dar comienzo a la ejecución del algoritmo manual. Hasta que el usuario no pulse "Aceptar" no se realizará ninguna acción.

Una vez que el usuario pulse "Aceptar" lo único que tiene que hacer es esperar a que termine el proceso. Por la apariencia de la aplicación parece que al pulsar "Aceptar" la aplicación no ha realizado absolutamente nada, sin embargo, internamente está realizando todas las asignaciones entre profesor y oferta que se han indicado manualmente, solo que dicho proceso permanece oculto al usuario.

Al finalizar se nos mostrará el siguiente mensaje indicándonos los archivos que han sido modificados, cuyo contenido puede consultarse en el apartado "*7. FICHEROS RESULTANTES*".

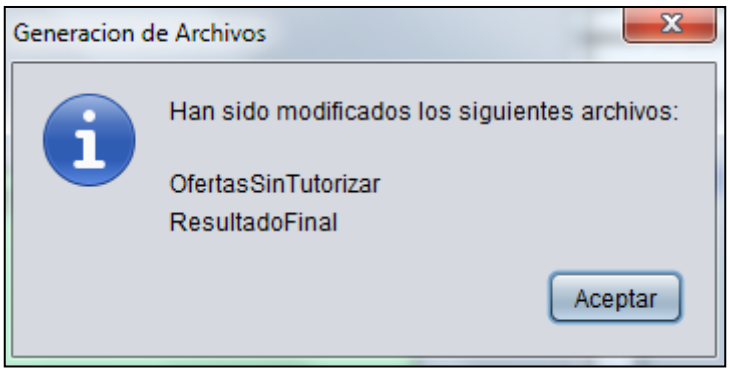


Fig. 55. Mensaje informativo. Lista de archivos que han sido modificados.

7. FICHEROS RESULTANTES

- **Alumnos de Intercambio.**

  Lista de alumnos que a la hora de leer los archivos de las matriculas se ha descubierto que no están matriculados en uno de los 3 grados disponibles (G420, G430 y G421). No serán tenidos en cuenta para el proceso.

Archivos de la primera parte **ALUMNO-OFERTA**

- **OfertasSinAsignar.xlsx**

  Lista de ofertas con el flag Asignada=’N’, es decir, ofertas que han publicado los centros que todavía no tienen un alumno para cursarlas.

- **OfertasAsignadas.xlsx**

  Lista de ofertas con el flag Asignada=’S’, es decir, ofertas que ya se encuentran asignadas con un alumno en concreto.

- **PI_AlumnosSinAsignar.xlsx**

  Lista de alumnos matriculados en alguno de los Practicum I de alguno de los grados, y que todavía no han sido asignados a ninguna oferta, es decir, ninguna de sus posibles elecciones en el formulario se ha podido cumplir.

- **PII_AlumnosSinAsignar.xlsx**

  Mismo caso que PI_AlumnosSinAsignar.xlsx pero para los alumnos de alguno de los Practicum II.

- **PIII_AlumnosSinAsignar.xlsx**

  Mismo caso que los anteriores pero para los alumnos de alguno de los Practicum III.

Archivos de la segunda parte **PROFESOR-ASIGNACIÓN**

- **OfertasSinTutorizar.xlsx**

  Lista de ofertas presentes en la tabla asignación con el flag Tutorizada=’N’, es decir, son ofertas que tienen un alumno asignado ya para cursarlas, pero que todavía no se encuentran tutorizadas por ningún profesor.

- **ResultadoFinal.xlsx**

  Representa el resultado final de la ejecución del programa, es decir, aquella lista de ofertas que han sido asignadas a un alumno en concreto, y que además ya se encuentran tutorizadas por un profesor específico.

Anexo VI – Autorización del Autor para la publicación electrónica

LICENCIA

(Aprobada en Consejo de Gobierno el 18 de Diciembre de 2008)

"E-ciencia" es un proyecto enmarcado en el contrato programa de cooperación interbibliotecaria entre la Comunidad de Madrid y el Consorcio Madroño, para crear una plataforma digital de acceso libre y abierto a la producción científica en la Comunidad de Madrid, con la participación de las universidades públicas y de los centros de investigación en este ámbito.

El autor/a D/ña. ..... *(con*

DNI) ..... *, adscrito/a a la Universidad de* ..... *como* .....

DECLARA que es el titular de los derechos de propiedad intelectual, objeto de la presente

cesión, en relación con la obra " ..... *":*

que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual como titular único o cotitular de la obra.

En caso de ser cotitular, el autor declara asimismo que cuenta con el consentimiento de los restantes titulares para hacer la presente cesión.

O en caso de previa cesión a terceros de derechos de explotación de la obra, el autor declara que tiene la oportuna autorización de dichos titulares de derechos a los fines de esta cesión o bien que retiene la facultad de ceder estos derechos en la forma prevista en la presente cesión.
Con el fin de dar la máxima difusión a mi obra citada, a través de este repositorio institucional, ***CEDO*** *a la Universidad de Alcalá, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, para que pueda ser utilizada de forma libre y gratuita por todos los usuarios del repositorio y del portal e-ciencia, los derechos de reproducción, de distribución, de comunicación pública, incluido el derecho de puesta a disposición electrónica, y el de transformación.*

La cesión se realiza en las siguientes condiciones:

El repositorio no asume la titularidad de la obra, que sigue correspondiendo al autor. Sin embargo, esta cesión de derechos sobre la obra permitirá al repositorio:

(a) Transformarla, en la medida en que ello sea necesario, para adaptarla a cualquier tecnología susceptible de incorporación a Internet; realizar las adaptaciones necesarias para hacer posible la utilización de la obra en formatos electrónicos, así como incorporar los metadatos necesarios para realizar el registro de la obra, e incorporar también "marcas de agua" o cualquier otro sistema de seguridad o de protección.

(b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores a los efectos de seguridad, de conservación, y de preservación del formato.

(c) Distribuir a los usuarios copias electrónicas de la obra en un soporte digital.

(d) Su comunicación pública y su puesta a disposición a través de un archivo abierto institucional, accesible de modo libre y gratuito a través de Internet.

La cesión es no-exclusiva, por lo que el autor es libre de comunicar y dar publicidad a la obra, en esta y en posteriores versiones, a través de los medios que estime oportunos.

El autor garantiza que el compromiso que aquí adquiere no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
El autor garantiza asimismo que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.

El autor asume la responsabilidad para el caso de que la Universidad de Alcalá fuera condenada por infracción de derechos derivada de las obras objeto de la cesión.

El autor, como garante de la autoría de las obras, asume toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, con fines de estudio, investigación, o cualquier otro fin lícito. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.

La Universidad deberá informar a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente.

La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusiva del autor.

La Universidad no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras.

El autor podrá solicitar la retirada de la obra del repositorio por causa justificada. A tal fin deberá ponerse en contacto con el responsable de e_Buah (ebuah@uah.es). Asimismo, el repositorio podrá retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

El autor será convenientemente notificado de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

En Alcalá de Henares, a ..... de ..... de 20..... .

POR LA UNIVERSIDAD
Excma. Sra. María Luisa Marina Alegre
Vicerrectora de Investigación de la UAH

POR EL AUTOR/A
Don/Doña